Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Luka Golinar

Sistem za oddaljeni dostop do merilnih naprav Red Pitaya

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: viš. pred. dr. Robert Rozman

Ljubljana, 2016

Fakulteta za računalništvo in informatiko podpira javno dostopnost znanstvenih, strokovnih in razvojnih rezultatov. Zato priporoča objavo dela pod katero od licenc, ki omogočajo prosto razširjanje diplomskega dela in/ali možnost nadaljne proste uporabe dela. Ena izmed možnosti je izdaja diplomskega dela pod katero od Creative Commons licenc http://creativecommons.si

Morebitno pripadajočo programsko kodo praviloma objavite pod, denimo, licenco *GNU General Public License, različica 3.* Podrobnosti licence so dostopne na spletni strani http://www.gnu.org/licenses/.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo: Sistem za oddaljeni dostop do merilnih naprav Red Pitaya.

Tematika naloge:

Izdelajte zasnovo strežnika za oddaljeni dostop do skupine merilnih sistemov Red Pitaya. Pri tem izkoristite vse možnosti oddaljenega dostopa, ki so že vsebovane v teh merilnih sistemih. Po potrebi dopolnite manjkajoče dele in oblikujte rešitev, ki poleg celovitega dostopa na daljavo, omogoča tudi upravljanje z uporabniki in dodeljevanje ustreznih pravic dostopa. Sistem preizkusite tudi v praksi na primeru izdelave lastne aplikacije za merilne sisteme Red Pitaya.

Rad bi se zahvalil vsem članom družine, ki so me podpirali v študijskih letih in sošolcem, s katerimi smo se skupaj učili pozno v noč. Zahvala gre tudi šestemu štuku in vsem kolegom za podporo v najhujših dneh. Zahvalil bi se rad tudi mentorju, Robertu Rozmanu, ki si je v veliki časovni stiski vzel čas zame in mi bil v veliko pomoč. Posebej pa se zahvaljujem tudi punci Urški za ves njen trudi in podporo takrat, ko sem jo najbolj potreboval. Brez tebe mi verjetno tega dela, vsaj v taki obliki, ne bi uspelo narediti.

Svoji dragi Urški.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
ARM	Advanced RISC Machine	Napredna RISC naprava
DAC	Digital to analog converter	Digitalno analogni pretvornik
ADC	Analog to digital converter	Analogno digitalni pretvornik
FPGA	Field programmable gate array	Programirljivo logično polje
SPA	Single page application	Spletna aplikacija z eno stranjo
SCPI	Standard commands for programmable instruments	Standardni ukazi za programirljive instrumente

Povzetek

Naslov: Sistem za oddaljeni dostop do merilnih naprav Red Pitaya

V diplomskem delu smo podrobneje predstavili merilni sistem Red Pitaya z nekaterimi osnovnimi orodji. Razvili smo sistem za oddaljen dostop, ki omogoča povezovanje, kontrolo in rezervacijo Red Pitaya merilnih sistemov na nekem oddaljenem strežniku. Vključili smo obstoječi SCPI strežnik v sistem za oddaljen dostop in dodali možnost pisanja kratkih SCPI programov. Razvili smo program za dostopanje do FPGA registrov merilnega sistema Red Pitaya in ga vključili v sistem za oddaljen dostop. Dodali smo možnost shranjevanja podatkov o signalih iz obstoječih aplikacij za napravo Red Pitaya. Za ta namen smo razvili namensko aplikacijo, ki temelji na ogrodju obstoječe aplikacije Generatorja in Osciloskopa. V aplikaciji smo dodali nekaj osnovnih funkcionalnosti, kot je vklop LED diod in shranjevanja podatkov na sistem za oddaljen dostop. Sistem ima možnost pregleda ter brisanja shranjenih podatkov. Sistem med drugim omogoča tudi dodajanje merilnih sistemov Red Pitaya, dodajanja uporabnikov in spreminjanja osebnih podatkov, s katerimi se prijavimo v sistem. Za konec smo prikazali način, kako lahko z uporabo strežnika sistem za oddaljen dostop objavimo na spletu.

Ključne besede: Red Pitaya, sistem za oddaljen dostop, spletne aplikacije Red Pitaya, podatkovna baza, SCPI strežnik, FPGA, registri.

Abstract

Title: The Remote Access Server for Red Pitaya Measurement Devices

In this thesis, we took a closer look at the Red Pitaya measurement system with some of its basic tools. We developed a Remote Access Server for access, reservation and control of Red Pitaya systems. We integrated the existing Red Pitaya's SCPI server into the remote access server and added a feature for writing small SCPI compatible programs. We developed a program for accessing FPGA registers on the Red Pitaya, and integrated it with our remote access server. In the scope of this work, we also created a simple Red Pitaya application on the base of existing Red Pitaya's Oscilloscope and Generator frameworks. We added a basic feature for turning on and off LED diodes, and a feature for saving data about acquired signals, which can be viewed and deleted remotely. The server also supports adding users and Red Pitaya measurement systems to the database. At the end, we explained how to use Nginx to deploy our application.

Keywords: Red Pitaya, remote access server, Red Pitaya web application, database, SCPI server, FPGA, registers.

Poglavje 1

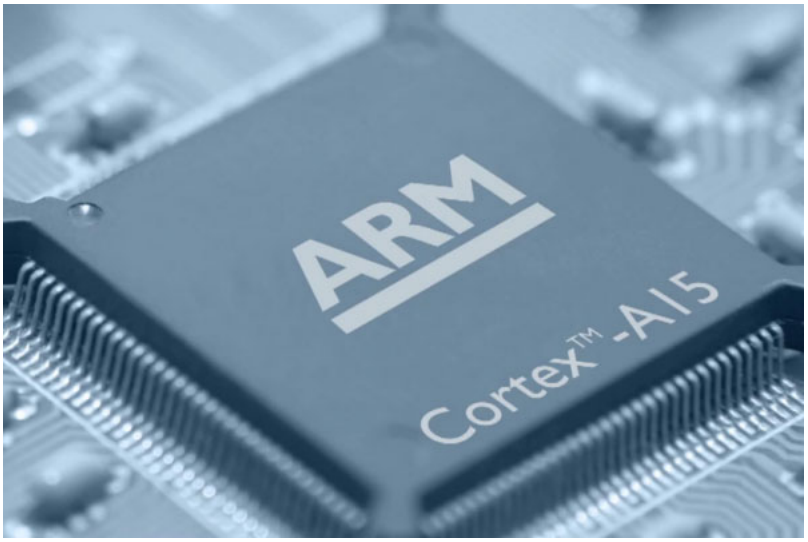
Uvod

Vse več računalniških storitev se dandanes dogaja v oblaku. Fizični računalniki se vse bolj pojavljajo v obliki strežnikov, ki shranjujejo na tisoče podatkov, stacionarne računalnike pa so nadomestili prenosniki in pametni telefoni. Storitve, kot na primer Amazon Host Service ter Microsoftov Azure, omogočajo širok nabor plačljivih storitev (virtualizacijo strežnikov, shranjevanje podatkov, virtualne podatkovne baze ...), do katerih lahko dostopamo praktično od vsepovsod.

Na drugi strani so naprave z vgrajenim ARM procesorjem. ARM (Advanced RISC Architecture) je 32-bitna arhitektura RISC razvijalca z istim imenom. Danes se uporablja v vgrajenih sistemih zaradi majhne električne porabe. Uporaba ARM procesorja prevladuje na trgu prenosnih naprav, kjer je nizka električna poraba pomemben kriterij razvijalcev [1]. ARM arhitekturo so prvič razvili leta 1983 v razvojnem projektu podjetja Acorn Computers Ltd s ciljem razvoja kompaktnega procesorja RISC. Razvojna ekipa je prvo razvojno različico, imenovano ARM1, končala aprila 1985, prvi proizvodni sistem, ARM2, pa je nastal naslednje leto. ARM2 je vključeval 32-bitno podatkovno vodilo, 26-bitni pomnilniški prostor in šestnajst 32-bitnih registrov. Danes se arhitektura ARM uporablja v približno 75% vseh 32-bitnih procesorjev RISC.

Idejo za svojo diplomsko nalogo sem dobil ob prvem srečanju z merilnim

sistemom Red Pitaya. Podjetje Red Pitaya oglašuje napravo kot prvi pravi prenosni merilni instrument in to idejo smo v diplomski nalogi poskušali še nadgraditi. Vse več instrumentov in merilnih naprav namreč omogoča oddaljen dostop preko računalnikov in mobilnih naprav, globalno združevanje podatkov na enem mestu pa postaja vse bolj in bolj popularen trend (vsak lahko, na primer, dostopa do IBM-ovega superračunalnika in na njem izvaja operacije). Na instrumentu Red Pitaya sem se zato odločil postaviti preprost sistem za oddaljen dostop in nadzor merilne naprave. Od tu dalje ga bomo imenovali RAS.



Slika 1.1: Primer ARM Cortex A15 procesorja [2]

V nalogi bomo podrobneje pogledali delovanje merilnega sistema Red Pitaya in zasnovali preprost sistem za rezervacijo in upravljanje ene ali več naprav Red Pitaya (RAS). Zgradili bomo tudi preprosto namensko aplikacijo, kompatibilno s sistemom Red Pitaya; predstavljala bo način, kako Red Pitaya aplikacija komunicira z RAS sistemom. Z aplikacijo bomo pokazali, kako lahko na preprost način uporabimo ogrodje obstoječih odprtih aplikacij in za naše potrebe naredimo svojo. RAS sistem omogoča oddaljen dostop do aplikacij, ki so na voljo na merilnem sistemu Red Pitaya. Pri tem je zelo pomembna možnost, da si lahko vsak uporabnik zgradi svojo aplikacijo.

Pri izbiri tematike diplomske naloge smo imeli več idej. Ena izmed teh je bila poenostaviti gradnjo oziroma programiranje namenskih aplikacij merilnih sistemov Red Pitaya. Tukaj velja poudariti, da je Red Pitaya razvojni projekt in se še vedno nenehno spreminja. Namen podjetja Red Pitaya je narediti učinkovit merilni sistem, ki bi bil hkrati preprost za uporabo in bi uporabnikom omogočal dodajanje in programiranje novih aplikacij. Podjetje razvija nove tehnologije, ki bi uporabnikom omogočale preprostejše pisanje aplikacij, zato smo se v tem primeru odločili, da se bomo raje kot izdelavi kompleksnejših aplikacij posvetili izdelavi celovite rešitve za oddaljen dostop. Prednost merilne naprave Red Pitaya je zaradi svoje velikosti prenosljivost, oddaljen dostop do sistema pa je precej omejen. Do naprave lahko dostopamo, če smo znotraj istega podomrežja ali pa neposredno preko utp mrežnega kabla. Hkrati se lahko zgodi, da imamo na voljo več različnih naprav Red Pitaya, ki bi jih radi upravljali. RAS sistem bi združil fizične naprave v skupno množico, do katere lahko dostopamo ter jo iz oddaljene lokacije nadzorujemo.

Slika 1.2: Logo podjetja Pitaya [3]

Poglavje 2

Uporabljene programske tehnologije

V naslednjem poglavju si bomo ogledali tehnologije uporabljene pri izdelavi sistema RAS. Vlogo in specifične funkcije vsake od teh tehnologij bomo podrobneje predstavili v poglavju 4 - Implementacija Sistema za oddaljen dostop. V tem poglavju si bomo pogledali spletna orodja in programske jezike, ki se uporabljajo za spletne strani in aplikacije, prav tako pa nizkonivojski jezik C, ki se v veliki meri uporablja za pisanje Red Pitaya aplikacij.

2.1 AngularJS

Aplikacija za oddaljen dostop je v bistvu spletna aplikacija. Kot uporabniški vmesnik nam služi javascript knjižnica AngularJS. Za to knjižnico smo se odločili, ker je bila primarno namenjena izdelavi spletnih aplikacij z eno samo stranjo (angl. single page application - SPA). Potreba po osveževanju velike količine podatkov, vzdrževanje stanja in vezave enega dela strani na druge, so nujno potrebne funkcije, ki jih AngularJS vsebuje. Spletna stran je tako veliko kvalitetnejša od običajnih HTML/Javascript spletnih strani, hkrati pa omogoča nešteto dodatnih funkcionalnosti [4].

2.2 Flask

Spletna aplikacija za oddaljen dostop ne more delovati brez aplikacijskega strežnika. Tukaj smo se odločili za orodje Flask, ki uporablja programski jezik Python. Orodje je zelo podobno strežniškim orodjem Django in Webapp. Flask je t.i. mikroogrodje, saj ne potrebuje specifičnih orodij in knjižnic, podpira pa možnost dodajanja številnih razširitev, kot so ORM (Object Relational Mappers), razne validacije podatkov, upravljanje nalaganja datotek, številne avtentikacijske tehnologije in druge. Za to orodje sem se odločil, ker odlično služi našim potrebam in lepo deluje skupaj z AngularJS knjižnico na uporabniškem vmesniku [5].

2.3 PostgreSQL

Naša aplikacija potrebuje poleg uporabniškega vmesnika in aplikacijskega strežnika tudi podatkovno bazo za shranjevanje podatkov. Odločil sem se za uporabo podatkovne baze PostgreSQL. To je ORMDBS oziroma object-relation database management system. Poudarek baze je na mnogih razširitvah. Uporablja se tako za manjše kot za velike internetne aplikacije z mnogo uporabniki in hkratnimi dostopi. PostgreSQL razvija PostgreSQL Global Development Group, skupek različnih podjetij in posameznikov. Orodje tako še vedno ostaja odprtokodno [6].

2.4 Programski jezik C

Večina Red Pitaya aplikacij je napisanih v programskem jeziku C. C sta leta 1972 razvila Dennis Ritchie in Bell Labs. Čeprav se je v letih veliko spremenil, ga je leta 1989 standardiziral ANSI oziroma American National Standards Institute. To je tudi razlog, zakaj se današnjemu C-ju pravi tudi ANSI-C. Je eden najbolj popularnih in javno razširjenih programskih jezikov na svetu. Uporablja konstrukte, ki zelo dobro in učinkovito preslikujejo strojne ukaze.

To je tudi eden izmed razlogov, zakaj so ga uporabili pri načrtovanju merilne kartice Red Pitaya [7].

2.4.1 Make

Make je orodje za prevajanje kompleksnih aplikacij, napisanih v nizkonivojskih jezikih kot sta recimo C ter C++. Make je razširjeno orodje, prisotno pri skoraj vsakem večjem projektu. Celoten Red Pitaya ekosistem, prav tako kot same aplikacije, potrebujejo zaradi svoje kompleksnosti za uspešno prevajanje makefile datoteke, v katerih se v bistvu nahajajo navodila za prevajanje. Nekatere makefile datoteke vsebujejo navodila za prevajanje novih makefile datotek. Tako hierarhično zgradbo lahko opazimo tudi na Red Pitaya ekosistemu [8].

2.4.2 Nginx

Nginx spletni strežnik je odprtokodno orodje za HTTP, HTTPS, SMTP in podobne standarde. Nginx je alternativa popularnemu Apache strežniku. Razvijati ga je začel Igor Szsoev leta 2002 za potrebe ruskih spletnih strani z veliko prometa. Spletni strežnik Nginx odgovarja na zahteve spletnih strani tako, da servira podatke. V projektu ga uporabljamo na več delih in je tako za nas zelo pomemben. Same aplikacije na Red Pitayi uporabljajo Nginx za serviranje statične vsebine, RAS pa uporablja Nginx za preusmerjanje zunanjega prometa na spletno aplikacijo in uporabnikom vrača odgovore [9].

Poglavje 3

Opis merilnega sistema Red Pitaya

V tem poglavju si bomo pogledali osnovne lastnosti naprave Red Pitaya. Opisali bomo proces zagona naprave in podrobno predstavili arhitekturo Red Pitaya aplikacij. Predstavili bomo ekosistem Red Pitaya in opisali vse komponente in direktorije, ki se v njem nahajajo. Teorija v poglavju je nujno potrebna za razvoj aplikacije, ki jo bomo naredili v sklopu diplomske naloge.

3.1 Tehnične podrobnosti sistema Red Pitaya

Red Pitaya je merilna kartica, ki spominja na mini računalnik RaspberryPi, vendar za razliko od slednjega opravlja popolnoma druge naloge. RaspberryPi se je prvotno razvijal kot pripomoček za učenje, danes pa služi za nešteto različnih funkcionalnosti, Red Pitaya pa je bila v prvi vrsti razvita kot odprtokodni merilni instrument. Potrebo po napravah, kot so osciloskop, generator in spektralni analizator, lahko najdemo praktično povsod. Vsaka od teh naprav služi točno določeni funkciji, njihov prenos na različne lokacije pa je zaradi velikosti in teže precej problematičen; omeniti velja tudi ceno, ki je pri profesionalnih instrumentih pogosto precej visoka. Napreden elektrotehnik ali kdorkoli, ki želi kaj izmeriti, mora tako imeti na voljo široko

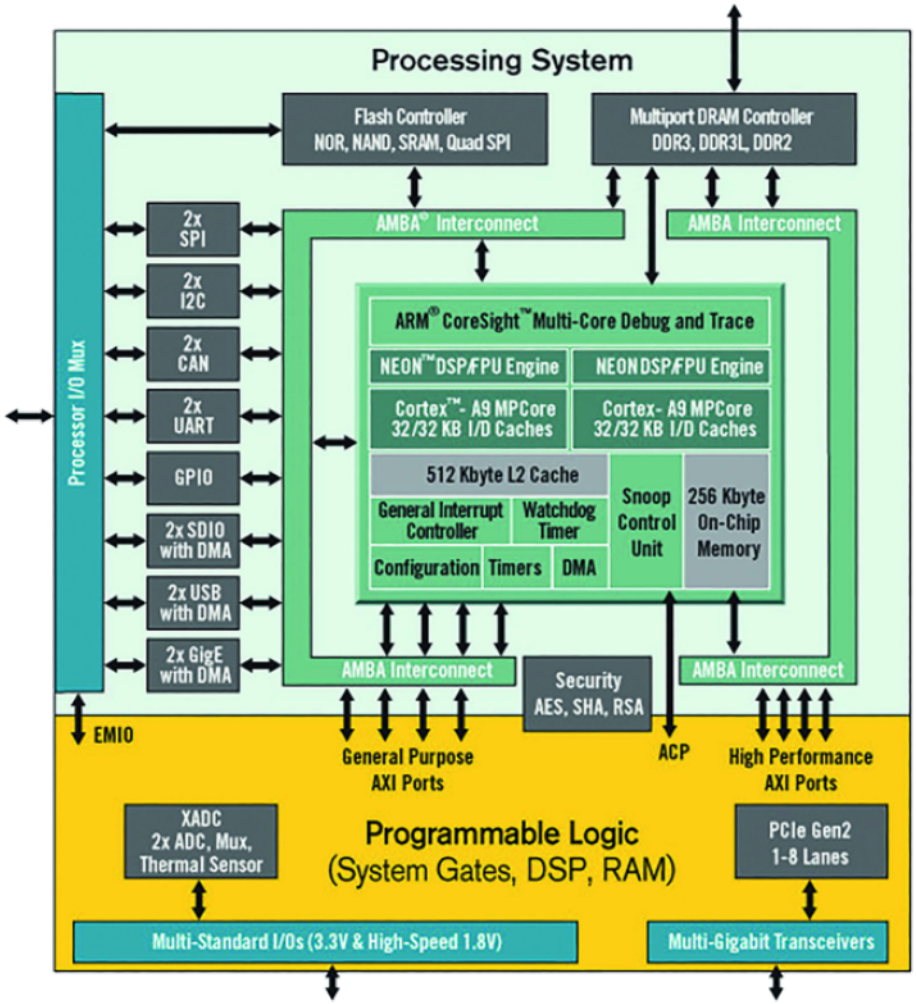
paleto merilnih naprav. Naprava, ki bi vse te inštrumente združila v enega, bi bila tako zelo dobrodošla. Koncept merilne naprave Red Pitaya se je pojavil leta 2013, ko je podjetje Instrumentation Technologies začelo s kampanjo na Kickstarterju, s pomočjo katere so predstavili majhno merilno kartico, ki bi v sebi imela zmogljivost in tehnologijo vseh dragih in velikih naprav, ki smo jih omenili. Po uspešnem zaključku projekta je kot izdelek podjetja Instrumentation Technologies nastala Red Pitaya,. V podjetju se je leta 2014 del ekipe odcepil in tako je nastalo samostojno podjetje Red Pitaya.

3.1.1 Arhitektura naprave Red Pitaya

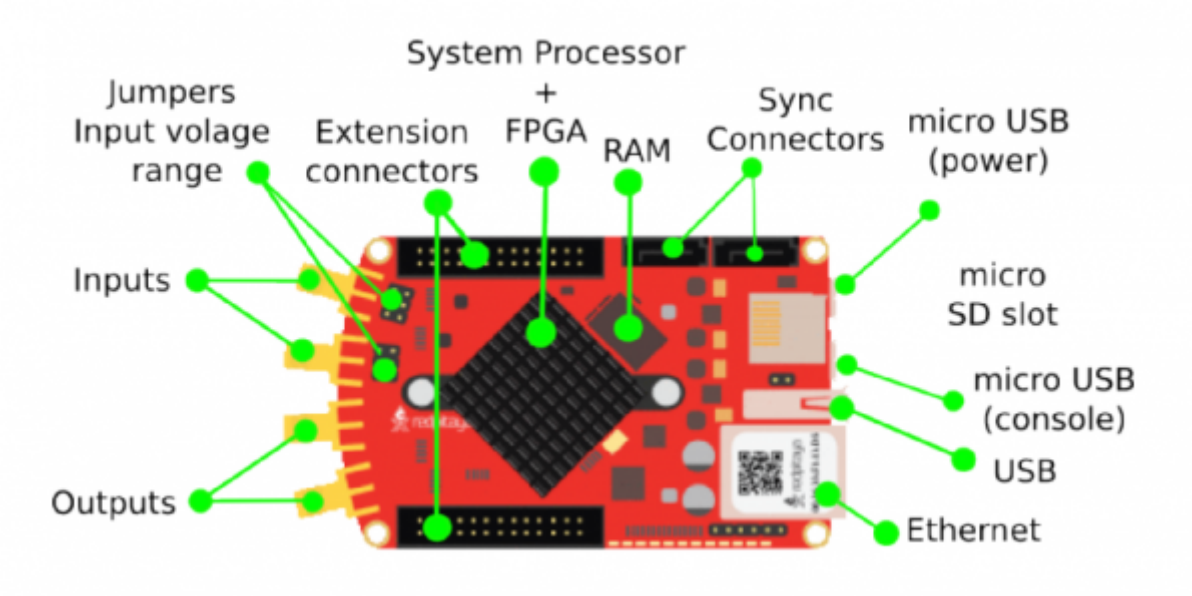
Srce naprave Red Pitaya predstavlja Xilinxov Zynq 7010 SoC čip, ki vsebuje dvojedrni ARM Cortex A9 procesor. Poleg samega procesorja imamo na voljo tudi 512 megabajtov DDR3 sistemskega spomina, v katerega lahko shranjujemo podatke. Družina Zynq-7000 procesorjev je bila namensko narejena za kombinacijo tako programskega kot FPGA programiranja, ki je sicer bolj tipično za FPGA platforme [11]. Tako kot pri ostalih merilnih sistemih ima tudi Red Pitaya na razpolago osnovne komponente kot sta USB naprava in nadvse pomemben ethernet priključek, ki je tudi eden glavnih akterjev pri sami komunikaciji merilnega sistema Red Pitaya. Naprava ima ob straneh vhodno-izhodne priključke, na spodnjem delu pa je prostor za micro sd kartico [12]. Slika 3.1 prikazuje arhitekturo čipa SoC Zynq 7010.

V preteklosti je Pitayo poganjal Buildroot ARM operacijski sistem, ki ga je kasneje zaradi aptitude programskega paketa in ostalih kompatibilnih paketov zamenjal Debian operacijski sistem. V spodnjem levem kotu so LED diode stanja. Teh je skupno 11, zadnje tri so uporabljene za prikazovanje stanja sistema. Zadnja, modra, prikazuje zaznavo operacijskega sistema. Zelena prikazuje pravilni zagon sistema, predzadnja rdeča prikazuje obremenjenost procesorske enote z uporabo preproste .sh skripte, ki se vzpostavi ob zagonu in s frekvenco srčnega utripa prikazuje obremenjenost procesorja. Vzorec vklapljanja in izklapljanja LED diode je podoben srčnemu utripu, variabilna hitrost pa prikazuje obremenjenost. LED dioda številka 8 je večnamenska

dioda, ki v trenutni verziji naprave Red Pitaya prikazuje pisanje oziroma branje na micro sd kartico. Primer komponent na napravi prikazuje slika 3.2.



Slika 3.1: Arhitektura čipa SoC Zynq 7010 [10]

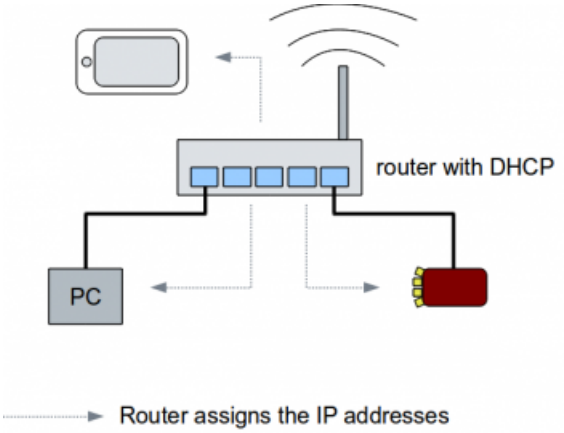


Slika 3.2: Prikaz osnovnih komponent na merilnem sistemu Red Pitaya [12]

3.1.2 Osnovni proces zagona merilne kartice

Red Pitaya ekosistem je zgrajen iz več komponent. Za izdelavo aplikacije za oddaljen dostop je potrebno dobro razumevanje sistema in številnih komponent, ki so potrebne za delovanje. Najprej si poglejmo osnovni proces zagona merilnega sistema Red Pitaya.

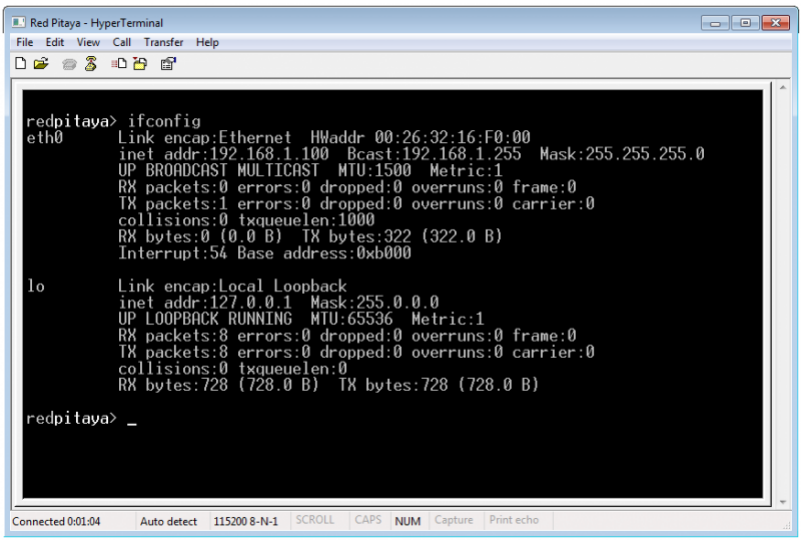
Uporabimo 5V napajanje z vsaj 1,5A toka in merilni sistem priključimo na priključek mikro usb, ki je namenjen za napajanje merilnega sistema. S tem začnemo proces zagona. Napravo lahko priključimo tudi na ethernet priključek. Ethernet linux gonilnik sistema Red Pitaya je nastavljen na DHCP; naprava tako pridobi IP naslov iz omrežja, v katerega je priključena. V primeru, da po dvajsetih sekundah ne dobi odgovora zunanjega DHCP strežnika, naprava samodejno nastavi IP naslov na 192.168.1.100 in mrežno masko na 255.255.255.0. Po uspešnemu zagonu lahko dostopamo do naprave preko IP naslova, ki nam ga je dodelil DHCP strežnik, ali pa preko samodejnega IP naslova.



Slika 3.3: Omrežna povezovalna shema Red Pitaya [13]

Na sliki 3.3 lahko vidimo, kako izgleda tipična povezava med merilnim sistemom Red Pitaya, usmerjevalnikom in uporabniškimi napravami. Tukaj velja omeniti, da je možno napravo, kot je računalnik, tudi neposredno priklopiti na merilni sistem. V tem primeru je potrebno ročno nastaviti IP naslov na 192.168.1.X in mrežno masko na 255.255.255.0

Kadar IP naslova ne vemo, lahko uporabimo serijsko konzolo in se preko UART protokola povežemo na napravo (Slika 3.4). Namestiti si je potrebno enega izmed programov za serijsko konzolo in nastaviti hitrost UART sprejemanja - BAUD RATE na 9200, ki je bitna hitrost UART protokola na napravi Red Pitaya.



Slika 3.4: Primer povezave na merilno kartico Red Pitaya preko serijske konzole [14]

Sam projekt Red Pitaya je še v veliki meri v fazi razvoja in lahko zelo hitro pride do kakšne napake, zato nam orodja za odpravljanje težav lahko pridejo še kako prav.

3.2 Arhitektura ekosistema Red Pitaya

V tem podpoglavju si bomo podrobneje pogledali sestavo ekosistema Red Pitaya. Poleg aplikacij najdemo v ekosistemu vse potrebne elemente za pravilno delovanje naprave.

3.2.1 Opis glavnih komponent v ekosistemu Red Pitaya

Red Pitaya ekosistem v bistvu predstavlja delovno okolje oziroma ogrodje. V samem direktoriju, ki je deloma na voljo tudi na spletnem portalu github [21], imamo več pomembnih komponent, ki zagotavljajo delovanje in celovitost sistema. Potrebno je omeniti, da se direktoriji in komponente v ekosistemu zelo hitro spreminjajo in bodo lahko v času branja te diplomske naloge nekateri drugačni, nekateri celo izbrisani, veliko jih bo morda dodanih. Opisali bomo nekatere najpomembnejše.

Direktorij	Kratek opis
Applications	V Applications direktoriju se nahajajo plačljive aplikacije. Te nas na tem mestu zaradi zaprtokodnosti ne bodo zanimale.
Bazaard/nginx	V tem direktorju se nahaja glavni komunikacijski sistem, ki povezuje kontroler trenutno aktivne aplikacije skupaj z njenim uporabniškim vmesnikom. Kontroler predstavlja knjižnico, v kateri se nahaja celotna logika trenutno zagnane aplikacije in je specifičen za vsako aplikacijo. Tega direktorija se ne bomo posebej dotikali, velja pa poudariti, da bo zelo prav prišel pri našem nadaljnjem delu na RAS.
Examples	Tukaj se nahajo primeri, ki prikazujejo nekatere praktične funkcionalnosti merilnega sistema Pitaya.
Os	Os direktorij vsebuje navodila za prevod jedra Linux. Samo jedro se prenese preko spleta, ob prevodu pa se dodajo še datoteke s spremembami obstoječe kode (angl. patch).

Test	Test je star direktorij, ki vsebuje nekatera orodja, uporabljena v starih ekosistemih. Uporablja se le še zaradi podpore starih sistemov, sicer pa ni več v uporabi.
Api/Api2	V direktoriju api in api2 se nahajajo API (Application programmable interface) funkcije, ki jih lahko vsak uporabnik uporabi pri izdelavi svojih aplikacij.
Apps-free	Apps-free direktorij vsebuje zastonjske aplikacije. Zaradi dostopnosti izvorne kode bomo za izdelavo naše testne aplikacije uporabili prav te.
Apps-tools	V apps-tools se nahajajo orodja za delo z ekosistemom. Do tega direktorija ne bomo dostopali.
Fpga/Fpga2	Prav tako lahko zelo na kratko opišemo direktorij fpga ter fpga2. Oba vsebujeta izvorno kodo za delovanje FPGA enote; Fpga2 predstavlja novejšo izdajo.
Patches	V direktorju patches se nahajajo vse spremembe oziroma dodatki, ki se aplicirajo ob prevajanju sistema (primer patcha oziroma spremembe smo omenili pri prevajanju Linuxovega kernela oziroma jedra.)
Scpi-server	V scpi-server direktoriju se nahaja še SCPI strežnik, katerega delovanje si bomo podrobneje pogledali v poglavju 4.2.
Tools	V tools direktoriju se nahajajo trenutno uporabljena orodja za delo z Red Pitayo.

3.2.2 Opis aplikacij Red Pitaya

Razumevanje ekosistema naprave Red Pitaya je pomemben dejavnik diplomske naloge. V tem poglavju si bomo na hitro pogledali nekaj najpomembnejših komponent in komunikacijskih kanalov, ki so za nas pomembni. Večji del poglavja bomo posvetili aplikacijam. Razdelimo jih lahko v dve večji kategoriji.

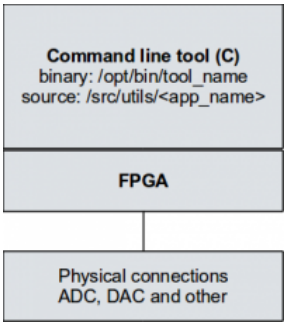
- Terminalske aplikacije

- Spletne aplikacije

Tako spletne kot terminalske aplikacije so za nas zelo pomembne, saj bomo v diplomskem delu potrebovali koncept obeh. Na kratko si bomo pogledali razlike med njima. Terminalske aplikacije so sestavljene iz dveh glavnih delov:

- Terminalskega orodja
- FPGA

Terminalsko orodje se nahaja v /opt/bin in do njega lahko dostopamo preko terminalske konzole. Za nas so terminalske aplikacije zelo pomembne, ker predstavljajo direktno povezavo z FPGA in kasneje strojno opremo. V nalogi si bomo tudi pogledali in izdelali preprosto terminalsko aplikacijo za komunikacijo z FPGA registri. Spodnja slika prikazuje komunikacijsko pot terminalskih aplikacij.



Slika 3.5: Tipična podatkovna pot terminalskih aplikacij Red Pitaya [15]

Za razliko od terminalskih aplikacij imajo spletne aplikacije vključeno še enoto komuniciranja in uporabniški vmesnik. Komponente, ki sestavljajo tipično Red Pitaya spletno aplikacijo, si sledijo v tej vrsti (od najvišjega do najnižjega nivoja):

- Končni uporabnik
- Grafični vmesnik
- Nginx

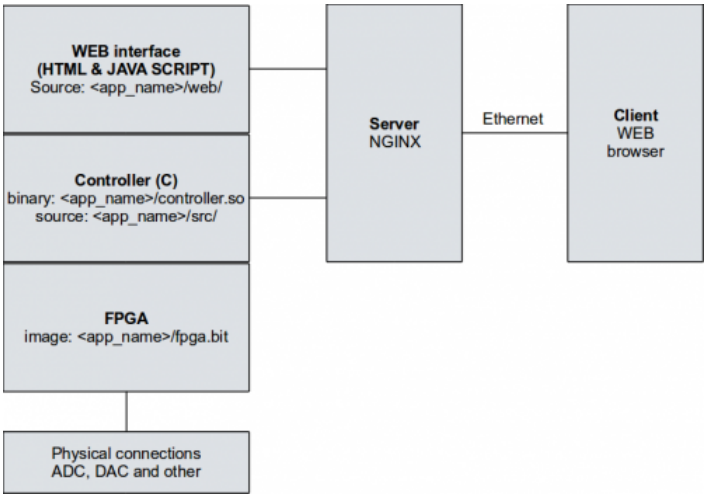
- Kontroler trenutno aktivne aplikacije
- FPGA

Za neizkušenega uporabnika predstavljajo spletne aplikacije glavnino merilnega sistema Red Pitaya. Večji del se torej dogaja v brskalniku. Končni uporabnik lahko do želene aplikacije dostopa preko brskalnika in IP naslova njegove naprave Red Pitaya. Ko uporabnik v brskalnik vtipka IP naslov naprave, se mu odpre okno, kjer se pojavi seznam vseh aplikacij, ki so trenutno na voljo. Ob kliku na želeno aplikacijo se odpre novo okno, kjer se izbrana aplikacija požene.

V ozadju se zgodi POST zahteva na Nginx strežnik na napravi Red Pitaya. Nginx strežnik dobi zahtevo in na podlagi imena servira statične datoteke za želeno aplikacijo. Za tem strežnik Nginx naloži delček programskega krmilnika želene aplikacije v spomin in preizkusi nekaj določenih funkcij, ki so značilne, nujne in potrebne za pravilno delovanje vsake aplikacije. Programski krmilnik predstavlja preprosto knjižnico, ki se prevede iz datotek programskega jezika C s končnico .c in .h v src direktoriju trenutne aplikacije. Po uspešnem testiranju Nginx naloži celoten programski krmilnik v spomin in uporabniški aplikaciji vrne prve podatke. Podatki, ki jih Nginx vrne uporabniku, so pravzaprav JSON paket, ki je sestavljen iz podatkov o signalih na obeh kanalih, ter iz parametrov. Slednji so uporabniške nastavitve določene aplikacije. Vzemimo aplikacijo scope+gen, ki je ena izmed aplikacij na napravi Red Pitaya in nam poleg osnovnih funkcionalnosti osciloskopa omogoča še generiranje signala. Izbiramo lahko tip signala, frekvenco, amplitudo, odmik in druge.

Slednji so edini način, s katerim lahko uporabniški vmesnik komunicira s programskim krmilnikom na Red Pitayi. Vezani so na sam krmilnik in so zrcalna slika parametrov, ki so definirani v njem; več o tem v naslednjem poglavju. Podatki se prenašajo preko paketa JSON, ki se zgradi v strežniku Nginx. Omeniti velja, da je funkcija strežnika Nginx tu zelo razširjena in kot smo videli, poleg samega posredovanja zahtev Nginx opravlja še veliko ostalih opravil, ki so bistvena za obstoj in delovanje Red Pitaya aplikacij.

Poleg signalov in parametrov vsebuje JSON paket še stanje povezave, ki jo dobi neposredno iz programskega krmilnika. Ta komunicira še z FPGA enoto, ki je v nalogi ne bomo uporabili. FPGA na koncu komunicira s strojno opremo.

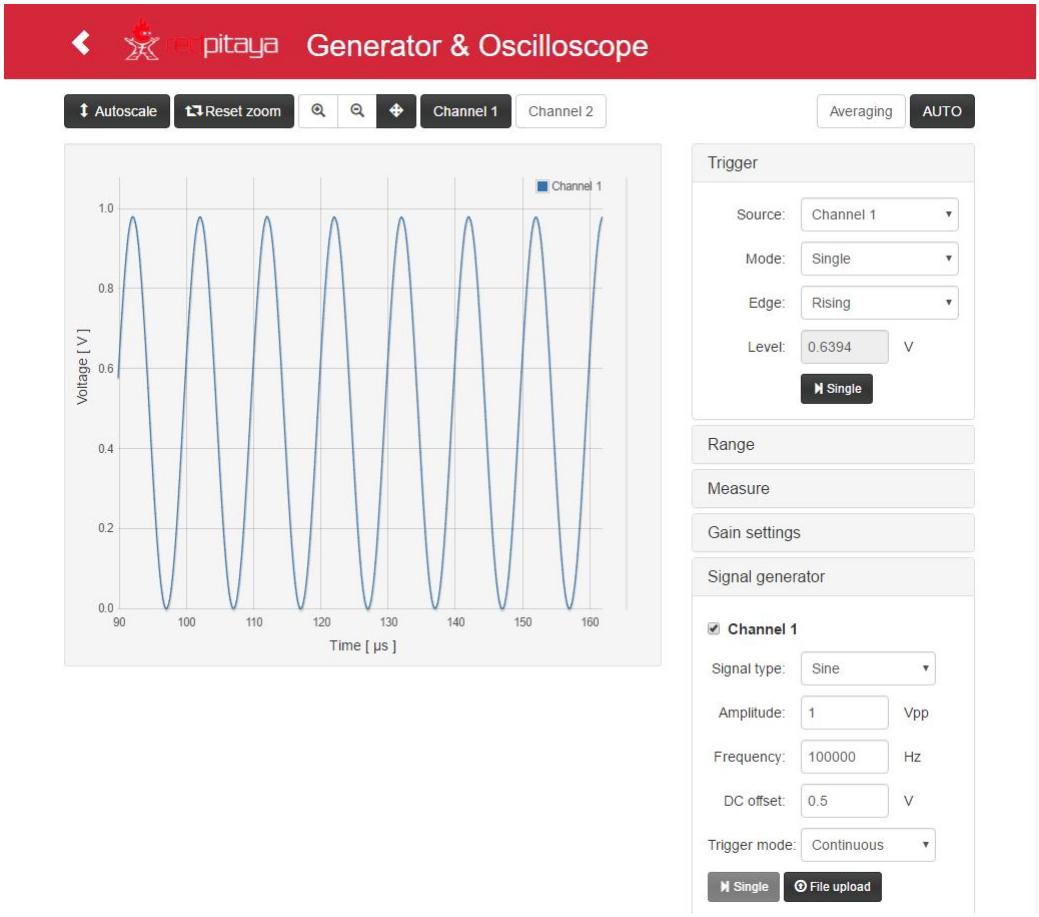


Slika 3.6: Prikaz tipične podatkovne poti spletnih aplikacij [16]

3.3 Osnovna arhitektura Red Pitaya aplikacij

V prejšnjem poglavju smo si pogledali dve vrsti aplikacij, ki jih poznamo na Red Pitayi, sedaj pa si bomo podrobneje pogledali strukturo aplikacije same. Obe poglavji sta za nas nadvse pomembni, saj se bomo v prihodnjih poglavjih ukvarjali s programiranjem tako spletnih kot terminalskih aplikacij. Pogledali si bomo osnovno strukturo aplikacije, njeno zgradbo, logični potek in delovanje. Naša testna primera bosta aplikaciji Osciloskop in Generator, ker vsebujeta tako zajem signala in uporabo enote ADC kot generiranje signala oziroma uporabo enote DAC.

Najprej si oglejmo samo strukturo aplikacije. Aplikacije najdemo v glavnem direktorju pod apps-free/. Ideja naprave Red Pitaya je bila od začetka odprtokodnost. Podjetje je podpiralo samoiniciativnost in razvoj svojih aplikacij. V zadnjem letu so precej obrnili smer razvoja in večina aplikacij je



Slika 3.7: Prikaz spletne strani aplikacije generator in osciloskop

sedaj zaprtih in plačljivih. Proste aplikacije se nahajo v direktorju apps-free. Te so za nas zelo pomembe, saj si lahko pogledamo, kako so zgrajene, jih spremenimo ali celo zgradimo svoje.

Vsaka aplikacija je zgrajena iz nekaterih pomembnih gradnikov. V tipični Red Pitaya aplikaciji se nahajajo naslednji direktoriji oziroma datoteke:

- doc
- info
- src
- Makefile

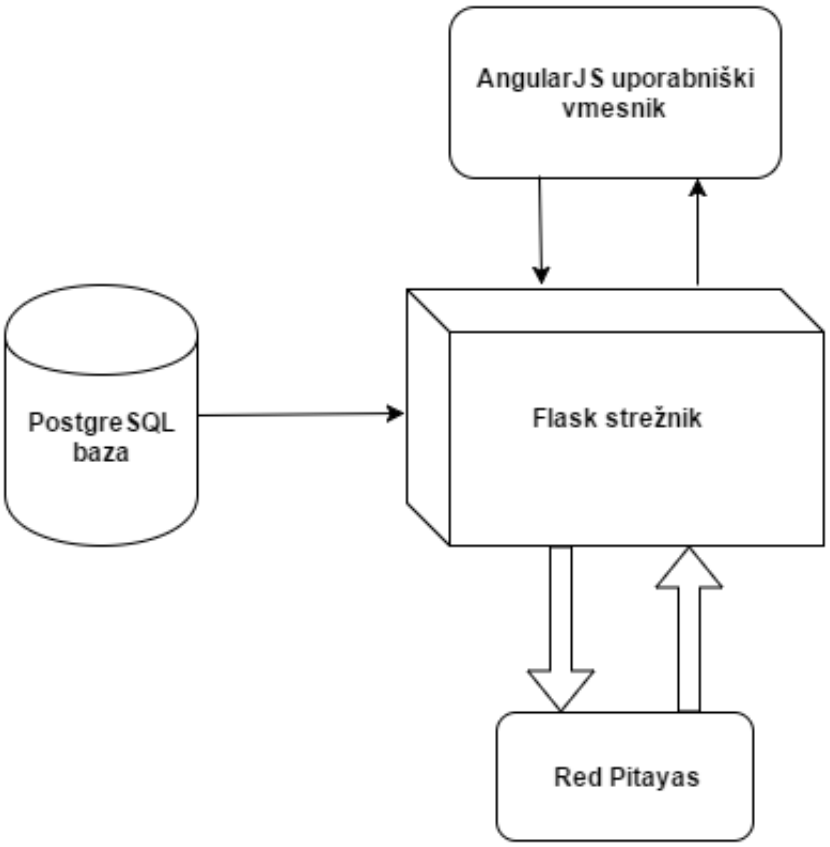
- fpga.conf
- index.html

V direktoriju doc najdemo dokumentacijo posamezne aplikacije. V info direktoriju se nahajajo info datoteke, ki so pomembne za prikaz osnovnih podatkov aplikacije na sistemu Red Pitaya. Datoteka Makefile se uporablja za prevajanje aplikacije. V njej je napisano navodilo, kako se mora vsaka aplikacija prevesti. Fpga.conf datoteka vsebuje pot do potrebne fpga.bit datoteke, konfiguracije FPGA dela merilnih sistemov Red Pitaya, ki je pomemben za komunikacijo z registri. Index.html je statična datoteka, ki se servira klientu kot uporabniški vmesnik. V direktoriju src se nahaja jedro aplikacije. Tukaj se skriva vsa logika (zajem, vzorčenje signala, branje iz strojnih registrov ipd.), ki je potrebna za pravilno delovanje aplikacije. Uporabniški vmesnik (index.html) o katerem smo govorili prej, načeloma samo prikazuje podatke.

Poglavje 4

Implementacija sistema za oddaljen dostop

Celotno strukturo sistema predstavlja glavni strežnik, na katerem teče Flask/AngularJS aplikacija. Strežnik je povezan v skupno podomrežje, na katerem se nahajajo tudi vse naprave Red Pitaya, ki so vključene v sistem. Konceptualni model prikazuje slika 4.1.

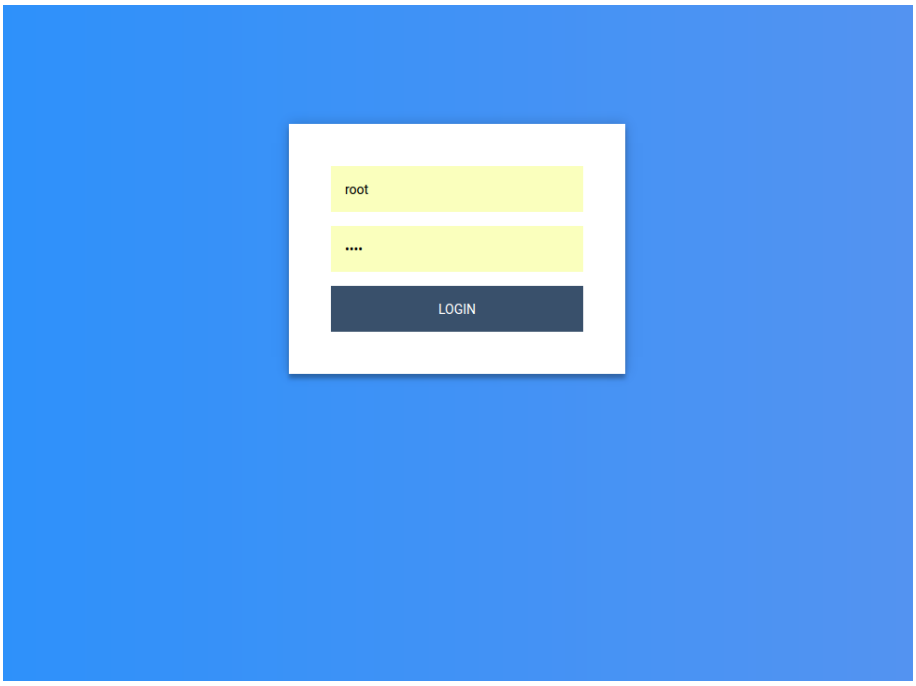


Slika 4.1: Prikaz povezav med ključnimi elementi RAS sistema

4.0.1 Vstopna spletna stran

Sistem za oddaljen dostop je implementiran z AngularJS in Flask tehnologijo. Uporabniški vmesnik je napisan v Angularju in sprejema uporabniške podatke, Flask server pa obdeluje podatke in komunicira z Red Pitayo. Ob navigaciji na naslov RAS sistema se nam pokaže preprosto vstopno okno. Prijavimo se lahko z uporabniškim imenom in geslom, ki je shranjeno v podatkovni bazi.

Ideja sistema je, da bi imel vsak uporabnik dostop do sistema in posledično do svojih naprav Red Pitaya.

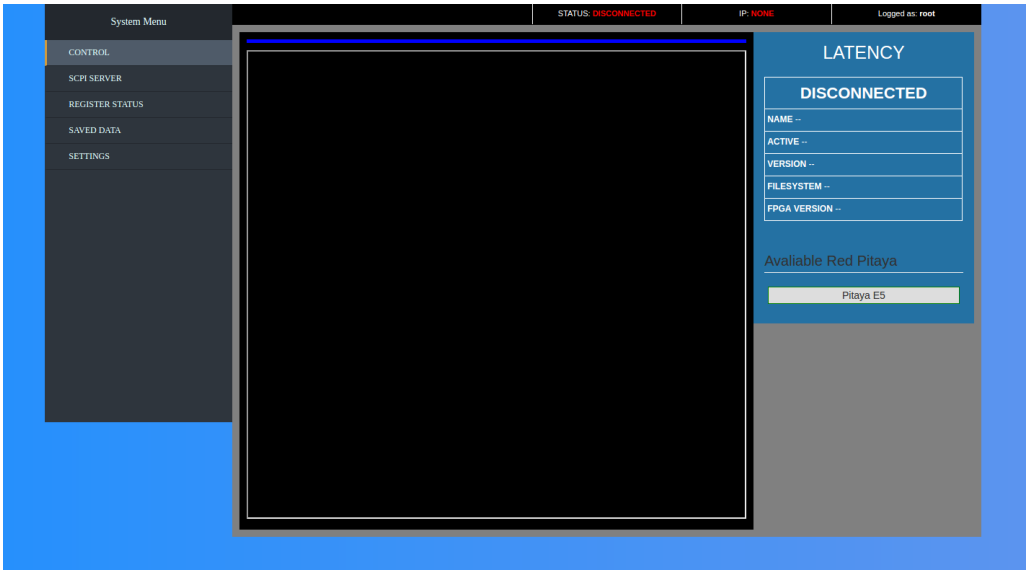


Slika 4.2: Tipična vstopna spletna stran v sistem RAS

4.1 Začetna spletna stran

Po prijavi v aplikacijo nas sistem poveže z RAS sistemom. Tukaj imamo poleg osnovnih podatkov, kot je menu, na voljo še nekaj osnovnih funkcionalnosti. Na zgornji strani imamo prikazane podatke o povezavi z merilnim sistemom Red Pitaya; na začetku je naše stanje seveda nepovezano. V desnem spodnjem oknu se nam prikažejo vse naprave Red Pitaya, ki so na voljo za povezavo. Vsak uporabnik ima poleg nekaj osnovnih podatkov v podatkovni bazi tudi določene naprave Red Pitaya, na katere se lahko poveže in z njimi upravlja. Vsaka naprava ima v bazi svoje polje, ki poleg imena in MAC naslova vsebuje tudi tuji ključ uporabnika, kateremu pripada. Sistem podpira več uporabnikov v istem okolju oziroma strežniku, vsak uporabnik pa nima nujno na voljo vseh možnih naprav. Sistem je tako kot nalašč za fakulteto, kjer lahko izvajalec dodaja merilne sisteme Red Pitaya, pripravi vaje, in potem vsem študentom omogoči dostop. Vsak uporabnik ima poleg

osnovnih lastnosti še stanje oziroma status.



Slika 4.3: Začetna spletna stran sistema RAS

4.1.1 Sistem za rezervacijo

V desnem spodnjem kotu se nam prikažejo merilni sistemi Red Pitaya, ki so na voljo za rezervacijo. Sistem deluje tako, da naredi podomrežni prelet vseh naslovov in izlušči iz njih vse naprave, ki imajo v imenu Instrumentation Technologies, ime, ki ga oddaja vsaka naprava Red Pitaya. Sistem nato preveri prijavljenega uporabnika in vse merilne sisteme, ki jih ima na voljo. Na podlagi IP naslova, gostiteljskega imena in samega uporabnika tako na začetno stran sistem izpiše vse naprave, ki so trenutnemu uporabniku na voljo (slika 4.3 prikazuje napravo Red Pitaya E5, ki nam je trenutno na voljo za povezavo). Za podomrežni prelet uporabljamo program *arp-scan*, ki nam je na voljo v Ubuntu Linux operacijskemu okolju, kjer se naš strežnik tudi izvaja.

```
rp_sweep = \
        subprocess.Popen(
            ['sudo', 'arp-scan',
             '--interface=wlp3s0',
             '192.168.1.0/24'],
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            stdin=subprocess.PIPE)
    stdout, stderr = rp_sweep.communicate()
```

Zgornji izsek kode nam nato vrne vse naslove, ki jih je našel v podomrežju v obliki niza. Niz potem razdelimo na vsak naslov posebej in iz njega dobimo vse naprave Red Pitaya, ki jih zapišemo na prvo stran. Na voljo imamo sicer tudi sistem za odkrivanje IP naslovov. Na spletni strani bazar.si lahko vnesemo MAC naslov želene naprave Red Pitaya in sistem nam pove, kakšen IP naslov ima. Idealni sistem za iskanje bi lahko uporabljal ta pristop, ampak na žalost že nekaj časa ne deluje in si moramo tako, dokler podjetje Red Pitaya napake ne odpravi, pomagati drugače.

4.1.2 Pridobivanje podatkov ter rezervacija oddaljene naprave

Sama rezervacija je precej preprosta. Uporabnik klikne na želen merilni sistem Red Pitaya, ki se mu v oknu pojavi v obliki gumba, sistem pa ga nato obvesti o uspešni oziroma neuspešni rezervaciji. Za samo rezervacijo se večinoma uporablja ukaz SSHFS (SSH Filesytem). Ukaz uporablja ssh in sftp protokol za namestitev oddaljenega sistema na želeno lokacijo. Vzpostavljeno drži sinhrono povezavo s sistemom in osveži direktorij, če se zgodi razlika na eni ali drugi strani. Sama namestitev se opravi v /tmp/IME-NAPRAVE-REDPITAYA direktoriju. Lokacija ni privzeta, ampak smo jo uporabili zaradi večnamembnosti /tmp direktorija, kajti po kliku na preki-

nitev povezave se vsebina direktorija pobriše.

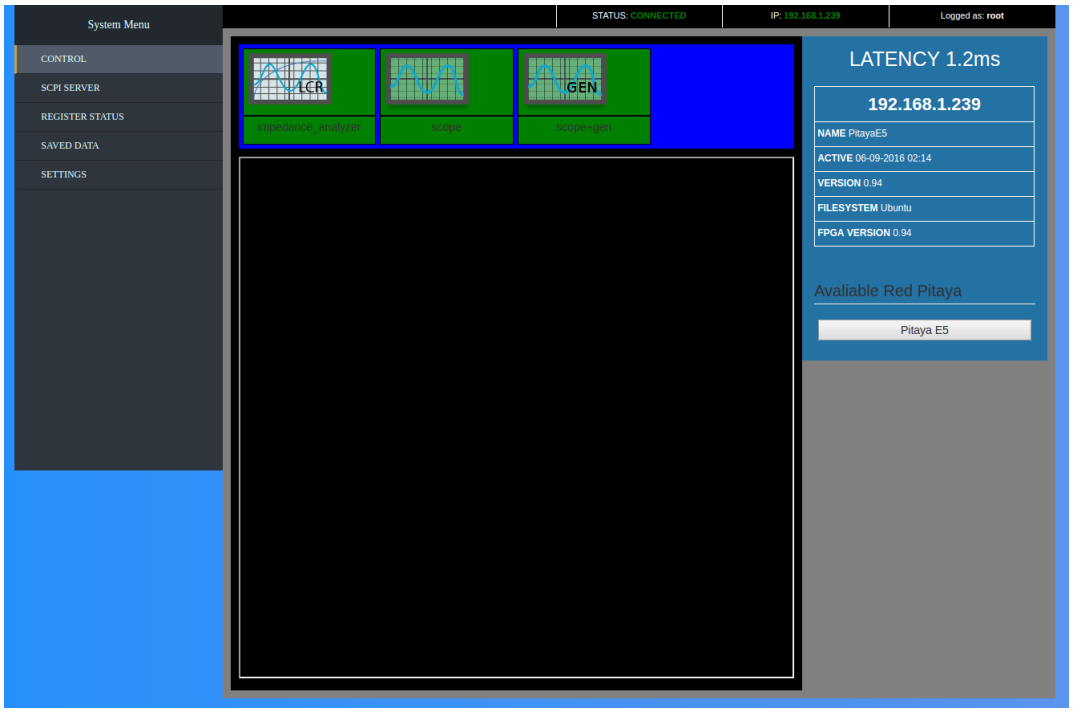
Aplikacija ob aktivni povezavi obdrži tudi določene informacije, ki prikazujejo trenutno povezan sistem. Med drugimi so tu podatki o IP naslovu, času aktivirane povezave, trenutna verzija datotečnega sistema na Pitayi in trenutna verzija FPGA sistema. Vsi ti podatki se prikažejo na RAS začetni strani.

Namen same aplikacije za rezervacijo je poleg amaterske tudi profesionalna uporaba, kot na primer kontrola oddaljenih sistemov in instrumentov. Morebitne zakasnitve tako lahko pri oddaji podatkov vplivajo na pravilno delovanje sistema RAS. V zgornjem desnem kotu je zato tudi predel, ki prikazuje hitrost oziroma zakasnitev povezave. Uporabniški vmesnik sistema RAS s časovnim zamikom sprašuje Flask strežnik o podatkih povezave. Ob inicializaciji povezave se na Flask strežnik strani ustvari nit, ki teče vse do konca same povezave in gostujočo napravo Red Pitaya nenehno sprašuje po ping paketih. Vsak paket poleg odgovora vrne tudi časovni zamik, ki ga prikažemo uporabniku na začetni strani.

Ob prekinitvi povezave oziroma ponovnemu kliku na gumb želene naprave Red Pitaya sistem sprosti povezavo z uporabo ukaza *fusermount*. Tako je pravkar sproščena naprava ponovno dostopna za rezervacijo.

4.1.3 Dostop do aplikacij na napravi

Na začetni strani lahko poleg osnovnih podatkov vidimo tudi aplikacije, ki so nam na merilnem sistemu na voljo. Aplikacije so prikazane s slikami in imenom, ki jih sistem pridobi iz info datotek posameznih aplikacij, ki se nahajajo v /info direktoriju. Aplikacije zaženemo tako, da kliknemo na ikono. Pojavi se vgrajeno okno, kjer lahko upravljamo z aplikacijo. Spodnja slika prikazuje primer aplikacij, ki so na voljo na merilnem sistemu Red Pitaya z imenom Pitaya E5.



Slika 4.4: Spletna stran z aplikacijami na oddaljeni napravi Red Pitaya

4.2 SCPI strežnik

Ideja diplomske naloge je oddaljen dostop do fizičnih naprav. V tem poglavju si bomo pogledali orodje, ki ga lahko uporabljamo tako na sistemu RAS kot zunaj njega in nam omogoča oddaljeno kontrolo instrumentov. SCPI orodje ima tako velik potencial pri nadzorovanju oddaljenih instrumentov.

4.2.1 Opis standarda

SCPI, kar je okrajšava za Standard Commands for Programmable Instruments, predstavlja večji del diplomske naloge. Standard definira sintakso in ukaze, ki se uporabljajo za merilne naprave in programabilne kontrolne teste. Leta 1990 je bil SCPI definiran kot dodatna plast na vrhu IEEE 488.2 specifikacije. Standard specificira skupno sintakso, ukazne strukture in podatkovne tipe, ki se morajo uporabiti v vseh merilnih instrumentih, ki pod-

pirajo ta standard. Vključuje tipične ukaze (npr. CONFigure, MEAsure), ki se lahko uporabljajo v vsakem SCPI instrumentu. Ukazi so razdeljeni v podskupine. SCPI definira tudi razrede instrumentov; tako ima, na primer, vsak instrument ki ga kontroliramo z PSU ukazom, isti DCPSUPPLY funkcionalni razred. Razredi instrumentov nam povedo, katere podrazrede je potrebno implementirati, prav tako pa tudi dodatne funkcionalnosti, ki so specifične za tovrsten instrument. Sama fizična povezava ni definirana znotraj SCPI standarda. Čeprav je bil SCPI standard narejen za IEEE-488, se lahko uporabi na kateremkoli komunikacijskem mediju, npr. Ethernetu, USB, VXIbusu in še mnogih drugih [18].

Slika 4.5: Prikaz IEEE-488 podatkovnega kabla [17]

4.2.2 Obvezni ukazi standarda IEEE 488.2

SCPI standard mora za skladnost z IEEE 488.2 specifikacijami zagotoviti nekatere obvezne ukaze. Vsi ukazi, tako splošni kot specifični za vsak instrument, so opisani z mnemoničnim imenom oziroma okrajšavo. Spodnja slika prikazuje vse obvezne ukaze, ki jih definira IEEE 488.2 standard [18].

V diplomski nalogi o posamičnih ukazih ne bomo pisali, ker niso predmet našega zanimanja. Več o ukazih si lahko preberete v PDF dokumentu o standardu IEEE-488.2; povezavo nanj najdete med literaturo [18].

Mnemonic	Name
*CLS	Clear Status Command
*ESE	Standard Event Status Enable Command
*ESE?	Standard Event Status Enable Query
*ESR?	Standard Event Status Register Query
*IDN?	Identification Query
*OPC	Operation Complete Command
*OPC?	Operation Complete Query
*RST	Reset Command
*SRE	Service Request Enable Command
*SRE?	Service Request Enable Query
*STB?	Read Status Byte Query
*TST?	Self-Test Query
*WAI	Wait-to-Continue Command

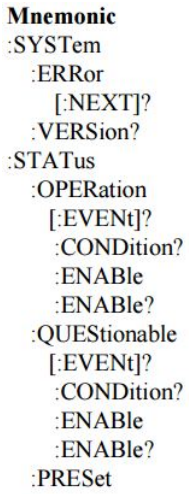
Slika 4.6: Obvezni IEEE-488 standardni ukazi [18]

4.2.3 SCPI Zahteve

Potrebno je omeniti, da IEEE 488.2 opisuje sintakso programiranja in obnašanje naprave le do neke mere. Dodatna izpopolnitev sintakse in dodatni ukazi so definirani v SCPI-99 standardu. Skupaj dajo napravi enoten "izgled in vtis". SCPI-99 standard poda tudi navodila in pomembne točke za izdelavo novih ukazov. Vsaka SCPI naprava mora torej poleg osnovnih IEEE-488.2 zahtev slediti še dodatnim, SCPI-99 zahtevam. Poleg potrebnih IEEE ukazov doda SCPI 99 standard še ukaze, ki so prikazani v spodnji sliki. V samem SCPI dokumentu imamo poleg zahtevanih ukazov opisane tudi opcijske ukaze, ki se lahko, na podlagi zmožnosti in značilnosti instrumenta, implementirajo ali ne [18].

4.2.4 SCPI Notacija

SCPI 99 standard poleg osnovnih ukazov definira tudi potrebno notacijo mnemoničnih imen ukazov. Spodnja tabela prikazujejo osnovno notacijo, hierarhična razmerja, parametre in asociacije. Razdeljena je na tri stolpce: KLJUČNA BESEDA, OBLIKA PARAMETRA in KOMENTARJI.



Slika 4.7: Obvezni ukazi SCPI-99 standarda [18]

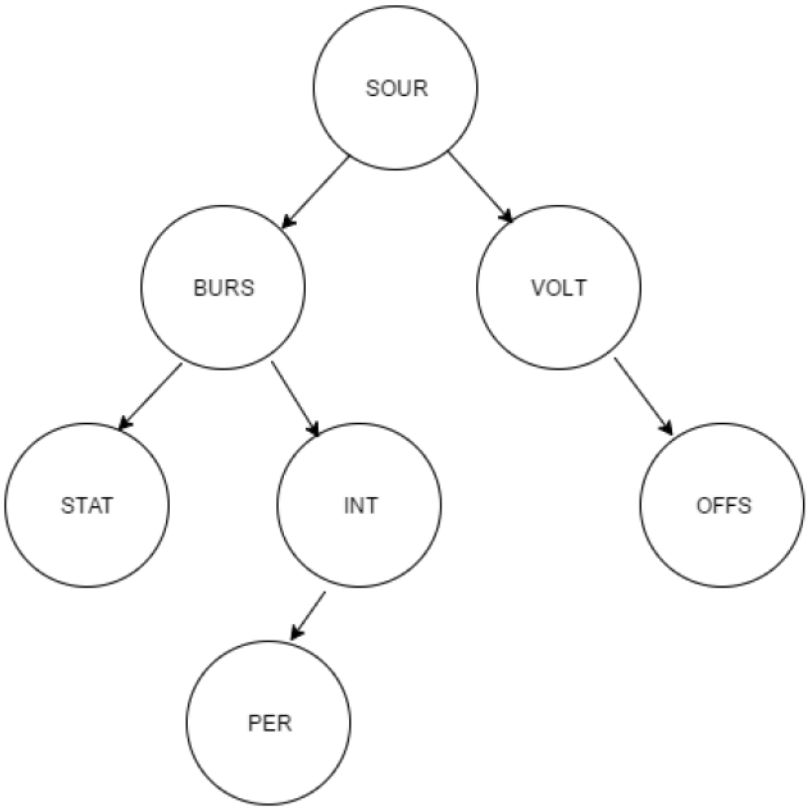
KLJUČNA BESEDA	OBLIKA PARAMETRA	KOMENTARJI
:FREQuency		
[:CW]	Numerična vrednost	
:AUTO	Spremenljivka tipa boolean	
:CENTer	Numerična vrednost	
:SPAN	Numerična vrednost	

Ključna beseda nam pove ime ukaza. Ker so SCPI ukazi bazirani na hierarhični strukturi, ki ji pravimo tudi drevesni sistem, je ime ukaza sestavljeno iz ene ali več ključnih besed. V takem sistemu so skupni ukazi združeni v skupno vozlišče. Predstavljamo si lahko binarno drevo, listi predstavljajo najnižjo hierarhično pozicijo. Višje kot gremo, bližje so vozlišča korenu, ampak vsako vozlišče lahko vodi do skupnih vej. V primeru, da želimo dobiti želen list ali pa želeno vejo, je potrebno podati celotno pot ukaza.

Oglati oklepaji ([]) se v KLJUČNI BESEDI uporabljajo za enkapsulacijo ključne besede, ki je opcijska, ko programiramo ukaz. Z drugimi besedami, naprava mora določen ukaz prepoznati kot istega, pa naj bo napisan v celoti, z vsebino znotraj oglatih oklepajev, ali pa brez nje. V PARAMETRU se oglati oklepaji uporabljajo za enkapsulacijo enega ali več parametrov, ki

so opcijski, zaviti oklepaji () pa se uporabljajo za enkapsulacijo enega ali več parametrov, ki so lahko vključeni enkrat ali nikoli. Ravna črta (—) se uporablja kot logični OR in lahko loči med enim ali več parametri. Ukazi se tipično delijo na zahteve in poizvedbe. Zahteva ima poleg samega ukaza na koncu še vprašaj in tipično vrne odgovor naprave.

Poglejmo si teorijo v praksi na uporabi preprostega ukaza za nastavljanje frekvence.



Slika 4.8: Primer drevesne strukture SOUR ukaza na SCPI strežniku Red Pitaya

Iz zgornjega drevesa lahko izpeljemo kar nekaj ukazov, primer dveh pa prikazuje spodnja koda.

```
SOUR1:VOLT 1
SOUR1:VOLT:OFFS 0.2
```

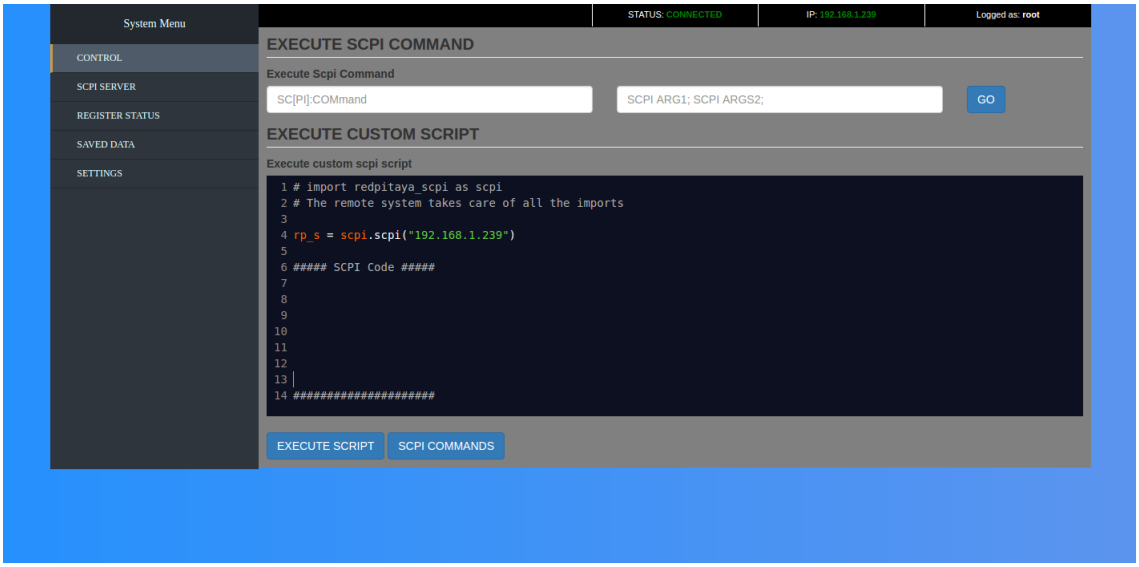
Z ukazoma nastavimo amplitudo in odmik generiranega signala. Ukazu SOUR smo podali tudi parameter, ki predstavlja številko kanala, na katerem želimo generirati signal.

4.2.5 Implementacija SCPI serverja na RAS

Tudi na sami Red Pitayi imamo implementiran SCPI strežnik. Strežnik na napravi Red Pitaya teče v obliki systemd storitve oziroma v obliki izvršljive datoteke. RAS komunicira s strežnikom z uporabo spletnih vtičev (angleško: sockets). Podpira izvršitev ukazov, ki so podprti na napravi Red Pitaya. Uporabnik vpiše ukaze v tekstovno polje in pritisne gumb za izvršitev. V primeru da je ukaz zahteva, RAS na spletno stran vrne odgovor strežnika. RAS sistem si tako lahko predstavljamo kot vmesni SCPI strežnik. V bistvu samo prebere ukaze, ki jih je uporabnik napisal in jih pošlje SCPI strežniku na napravi Red Pitaya.

4.2.6 Pisanje kratkih SCPI programov

RAS sistem omogoča tudi pisanje kratkih SCPI programov. Na uporabniški strani je implementiran sistem, ki podpira Python sintakso, v ozadju pa strežnik izvrši kodo, ki jo uporabnik napiše. V primeru napake le-to javi uporabniku. Sistem ne omogoča naprednih možnosti pisanja programov. V tem razdelku imamo tudi povezavo do dokumentacije podprtih SCPI ukazov na Red Pitayi. Ob kliku na gumb sistem uporabniku na računalnik prenese .odt dokument, kjer je napisana sintaksa vseh ukazov in možnih argumentov.



Slika 4.9: SCPI sekcija na sistemu RAS

4.3 Aplikacija za pridobivanje vrednosti registrov

Naslednji del spletne aplikacije se imenuje Register status. Na tej strani lahko aktivno spremljamo stanje vseh registrov na trenutno povezani Red Pitayi. Idejo smo dobili, ker se v razvojni fazi, pa tudi v vsakodnevni uporabi, večkrat pojavi potreba po pregledu stanja nekega registra. Za to imamo na Pitayi na voljo nekaj terminalskih orodij, kot je na primer monitor, ki nam omogočajo pregled stanja enega ali več registrov. Težave nastanejo, ker se je treba za aktivno preverjanje vedno prijaviti v terminalsko okolje. Sam proces je tako zamuden, hkrati pa se registri hitro spreminjajo. V ta namen smo razvili orodje, ki nam omogoča branje in prikazovanje registrov na naši spletni aplikaciji na pregleden in celovit način. Za boljše razumevanje orodja si bomo najprej pogledali nekaj ozadja o sami arhitekturi registrov in spominskih blokov v Red Pitayi.

Red Pitaya spominski prostor na FPGA je razdeljen na 8 blokov. Spodnja tabela prikazuje imena in začetne ter končne naslove vsakega izmed

posameznih blokov. Vsi registri imajo odmik štirih bajtov (4 bytes) in so široki 32 bitov, najmanjša velikost prenosa pa je 4 bajte. Biti so postavljeni v little-endian vrstnem redu. To pomeni, da je MSB (Most Significant Bit) oziroma najpomembnejši bit postavljen na skrajno desno stran, LSB (Least Significant Bit) oziroma najmanj pomemben bit pa je postavljen na skrajno desno stran 32-bitnega registra. Slika 4.10 je izrezana iz dokumenta na spletni strani Red Pitaya, kjer so znotraj spominskih blokov opisani tudi vsi registri, ki nimajo nujno istih lastnosti.

	Start	End	Module Name
CS[0]	0x40000000	0x400FFFFF	Housekeeping
CS[1]	0x40100000	0x401FFFFF	Oscilloscope
CS[2]	0x40200000	0x402FFFFF	Arbitrary signal generator (ASG)
CS[3]	0x40300000	0x403FFFFF	PID controller
CS[4]	0x40400000	0x404FFFFF	Analog mixed signals (AMS)
CS[5]	0x40500000	0x405FFFFF	Daisy chain
CS[6]	0x40600000	0x406FFFFF	FREE
CS[7]	0x40700000	0x407FFFFF	Power test

Slika 4.10: Prikaz razdelitve FPGA spominskega prostora na logične bloke [19]

Vsak register ima tako področja, v katera lahko pišemo, in taka, v katera ne moremo. Slika 4.11 prikazuje izsek registrov v t.i. Housekeeping spominskemu bloku. Vsak register je predstavljen z odmikom (offset), imenom in opisom bitov. Za primer lahko vzamemo LED control register. Odmik od osnovnega naslova je tukaj 0x30, bite od 31 do 8 lahko samo beremo, kar je razvidno iz oznake R (Read oziroma Branje), biti od 7 do 0 pa imajo oznako R/W (Read/Write oziroma Branje/Pisanje). Ker imamo na Red Pitayi Little Endian sistem, je tudi dokument sestavljen od MSB do LSB bita. Pri programiranju Pitaya aplikacij (tako spletnih kot terminalskih) je potrebno upoštevati, v katere bite lahko in v katere bite ne moremo pisati. Poglavje 3 opisuje osnovne gradnike, ki so potrebni za pravilno delovanje naprave Red Pitaya. V tem poglavju smo opisali smo tako terminalske kot spletne aplika-

Housekeeping

offset	description	bits	R/W
0x0	**ID**		
	Reserved	*31:4*	*R*
	Design ID 0-prototype0, 1-release1	**3:0**	**R**
0x4	**DNA part1**		
	DNA[31:0]	**31:0**	**R**
0x8	**DNA part2**		
	Reserved	*31:25*	*R*
	DNA[56:32]	**24:0**	**R**
0xC	**Digital Loopback**		
	Reserved	*31:1*	*R*
	Digital loop	**0**	**R/W**
0x10	**Expansion connector direction P**		
	Reserved	*31:8*	*R*
	Direction for P lines 1-out 0-in	**7:0**	**R/W**
0x14	**Expansion connector direction N**		
	Reserved	*31:8*	*R*
	Direction for N lines 1-out 0-in	**7:0**	**R/W**
0x18	**Expansion connector output P**		
	Reserved	*31:8*	*R*
	P pins output	**7:0**	**R/W**
0x1C	**Expansion connector output N**		
	Reserved	*31:8*	*R*
	N pins output	**7:0**	**R/W**
0x20	**Expansion connector input P**		
	Reserved	*31:8*	*R*
	P pins input	**7:0**	**R**
0x24	**Expansion connector input N**		
	Reserved	*31:8*	*R*
	N pins input	**7:0**	**R**
0x30	**LED control**		
	Reserved	*31:8*	*R*
	LEDs 7-0	**7:0**	**R/W**

Slika 4.11: Prikaz dela t.i. Housekeeping spominskega bloka v napravi Red Pitaya [19]

cije. Razlika med njimi je le v dodatku spletnega sistema, ki ga terminalske aplikacije, za razliko od spletnih, nimajo. Nižji nivo, tj. kontroler in FPGA, pa pri obeh vrstah aplikacij deluje na isti način. Pri samemu pisanju aplikacij se moramo poleg vse logike osredotočiti tudi na pisanje in branje v registre, ki pravzaprav predstavljajo zelo pomemben del pri sami komunikaciji z FPGA in strojno opremo.

4.3.1 Opis aplikacije za pridobivanje stanja registrov

Program za pridobivanje stanja registrov smo napisali v programskem jeziku C. Med opisanimi tehnologijami smo omenili, da je C zelo primeren za dostop in branje podatkov strojne opreme. Celoten programski krmilnik v aplikaciji Red Pitaya je napisan v jeziku C.

Program lahko razdelimo na tri večje dele oziroma komponente:

- Makefile
- Pomožni deli
- Glavni del

Za prevajanje programa smo tukaj uporabili make jezik, oziroma Makefile, ker nam je olajšal delo z prevajanjem. V podrobnosti Make jezika se ne bomo spuščali, slika 4.12 pa prikazuje odsek kode, ki skrbi za prevajanje .c datotek v datoteko, ki jo lahko poženemo na Red Pitayi.

```
#Cross compiler definition
CC = $(CROSS_COMPILE)gcc

CFLAGS = -g -std=gnu99 -Wall -Werror
OBJS := $(patsubst %.c, %.o, $(wildcard *.c))
EXECUTABLE = rp_registers

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJS)
	$(CC) -o $(EXECUTABLE) $(OBJS) $(CFLAGS)

clean:
	$(RM) *.o
	$(RM) $(OBJS)
	$(RM) $(EXECUTABLE)
```

Slika 4.12: Izsek Makefile datoteke za prevajanje naše aplikacije

Pomožni deli so sestavljeni iz .c in .h datotek, ki se uporabljajo za preslikovanje fizičnega FPGA spomina v logični prostor našega C programa. Vsak

spominski blok, ki smo ga opisali zgoraj, lahko priredimo enemu paru .c in .h datotek. Najprej smo definirali začetek naslovnega prostora in osnovno velikost za vsak spominski blok.

```
/* Global address and offset definition */
#define HK_BASE_ADDR 0x40000000
#define HK_BASE_SIZE 0x30000
```

To smo naredili v .h glavi C datoteke. Sam program uporablja C strukture, v katere se zapišejo vrednosti registrov. V .h datoteko vsakega spominskega bloka je tako potrebno definirati tudi strukturo, ki mora biti enaka zaporedju registrov, ki si sledijo v tem spominskem bloku. Za primer si ponovno oglejmo C strukturo za spominski blok t.i. Housekeeping, ki smo si ga ogledali že zgoraj.

```
/* Stucture definition */
typedef struct house_keep{
        uint32_t id; //0x0
        uint32_t dna_p1; //0x4
        uint32_t dna_p2; //0x8
        uint32_t exp_c_P; //0xC
        uint32_t reserved; //0x10
        uint32_t exp_c_N; //0x14
        uint32_t exp_out_P; //0x18
        uint32_t exp_out_N; //0x1C
        uint32_t exp_c_in_P; //0x20
        uint32_t exp_c_in_N; //0x24
        uint32_t reserved2; //0x28
        uint32_t reserved3; //0x2C
        uint32_t led_control; // 0x30
        // 0x30  [ 7: 1 RW | 0 RESERVED |
        // 7 > RESERVED]
} house_kp_t;
```

Potrebno je še poudariti, da je razlika med začetnim in končnim naslovom v t.i. Housekeeping tabeli precej večja, kot smo jo predstavili tukaj. Tipična praksa pri rezervaciji FPGA prostora je, da se zaradi morebitne opcije dodajanja registrov oziroma funkcionalnosti za vsak spominski blok rezervira več prostora, kot ga je potrebno. Opazimo lahko tudi nekaj polj, ki jih v sliki 4.11 ni. Tako kot pri skupnem prostoru se tudi med bloki lahko pusti prazen prostor za morebitne dodatne registre. Te registre je potrebno zaradi točnega C-preslikovanja v naši strukturi definirati, četudi jih ne bomo potrebovali.

Sedaj, ko imamo definirane vse potrebne parametre in nastavitve, je potrebno preslikanje fizičnih naslovov v logične, oziroma preprosteje: v definirano C strukturo je potrebno zapisati podatke, ki se nahajajo na zgoraj definiranih naslovnih prostorih. Za to nalogo smo uporabili metodo mmap, ki sprejme kar nekaj parametrov, vrne pa kazalec na blok spomina, kjer se nahajajo naši registri.

Velika prednost Linux operacijskega sistema je, da do vseh naprav dostopa preko datotek. Preko datoteke mem tako dostopa do spominskega prostora FPGA na naši napravi Red Pitaya. Za odpiranje uporabimo preprosto funkcijo open, kateri lahko poljubno dodamo tudi dodatne zastavice oziroma parametre.

```
int house_init(void){

        if(house_release() < -1){
                printf("Mapping resources failed.");
                return -1;
        }

        void *page_ptr;
        long page_addr, page_off, page_size = sysconf(_SC_PAGESIZE);

        /* OPEN THE DEVICE */
        house_fd = open("/dev/mem", O_RDWR | O_SYNC);

        if(house_fd  < -1){
                printf("Error opening /dev/mem\n");
                return -1;
        }

        /* Calculate page correct addresses */
        page_addr = HK_BASE_ADDR & (~(page_size - 1));

        page_off = HK_BASE_ADDR - page_addr;

        /* mmap physical memory */
        page_ptr = mmap(NULL, HK_BASE_SIZE, PROT_READ |
                PROT_WRITE, MAP_SHARED, house_fd, page_addr);

        if((void *)page_ptr == MAP_FAILED){
                printf("Mapping failed.\n");
                return -1;
        }

        house_reg = page_ptr + page_off;

        return 0;

}
```

Slika 4.13: Izsek kode za preslikovanje fizičnih registrov t.i. Housekeeping spominskega bloka

Koda v sliki 4.13 nam s pomočjo metode mmap vrne kazalec na začetek spominskega bloka. Začetek dejanskega bloka t.i. Housekeeping dobimo tako, da kazalcu, ki kaže na začetek strani, prištejemo še odmik "HOUSEKEEPING_BASE_ADDR", ki smo ga definirali v .h datoteki. Znotraj definiramo kazalec "housekeeping_control_t", kateremu nato priredimo vrednost, ki jo dobimo po prištetju odmika mmap metode. Tako na zelo preprost način preslikamo fizične registre v C strukturo. Enako ponovimo z vsemi ostalimi bloki. Vse, kar se razlikuje, so naslovi začetka bloka, oziroma odmik od začetka in velikost naslovnega prostora, ki ga zasede spominski blok.

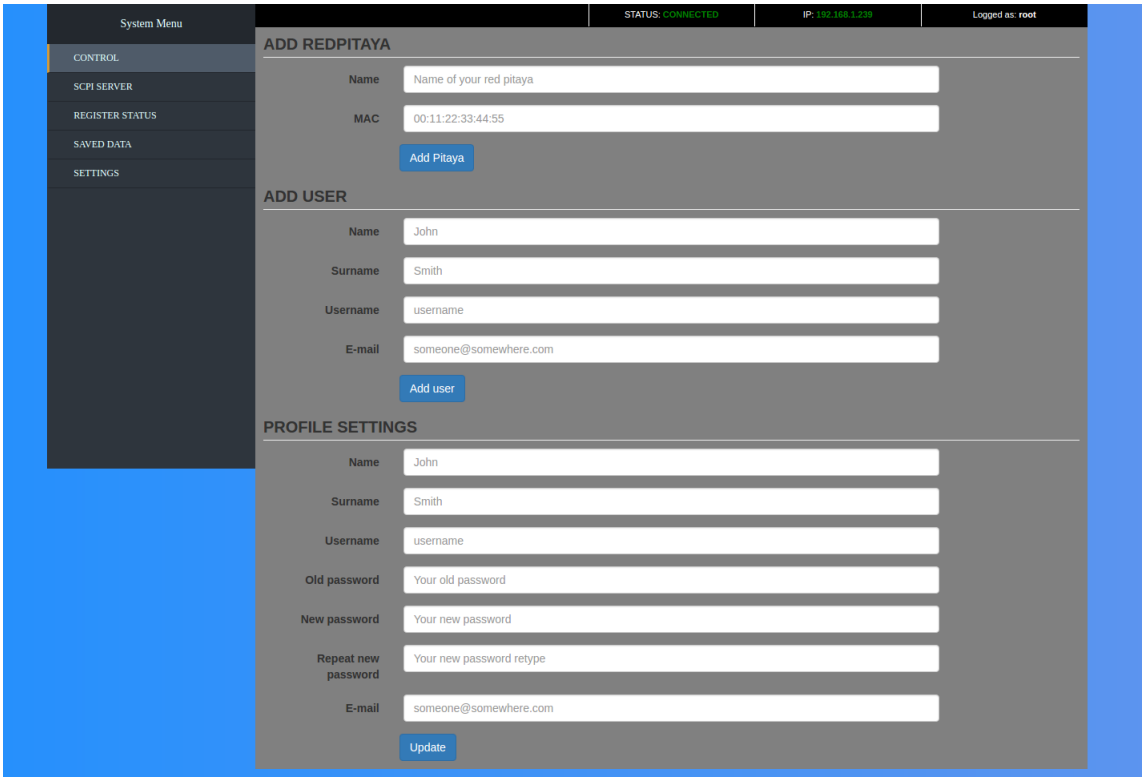
V pomožnih delih imamo poleg preslikovanja prostora tudi metodo, ki nam vrne kazalec na virtualni prostor, kjer se nahajajo podatki registrov. Te metode uporabimo v glavnem delu in tako na eno mesto dobimo vse podatke, ki so nam potrebni. Podatke nato zapišemo na datoteko na Pitayi. Razdelimo jih na kategorije, kjer prva vrstica predstavlja ime bloka, podatki, ki sledijo, pa predstavljajo ime registra in s presledkom ločeno vrednost v desetiški obliki.

Ob sami povezavi merilnega sistema Red Pitaya sistem za oddaljen dostop pošlje na želeno Pitayo še ukaz za zagon programa, ki smo ga napisali. Omenili smo, da SSHFS protokol obdrži sinhrono povezavo z oddaljenim in lokalnim sistemom. Tako lahko RAS datoteko, ki se ustvari na Pitayi, prebere in prikaže podatke. Podatke, ki jih dobimo iz C programa, zapišemo v "uint32_t"obliki, na RAS sistemu pa jih pretvorimo v binarno vrednost. Slika 4.14 prikazuje primer prikaza podatkov iz registrov na RAS. Poleg imena in desetiške oblike podatkov prikažemo še binarno. Na sami strani smo uporabili HTML gradnik tabbed pane za kategoriziranje oziroma ločitev registrov na spominske bloke.

Slika 4.14: Primer spominskega bloka t.i. Housekeeping na RAS sistemu

4.4 Osnovne nastavitve

V razdelku settings vidimo nekaj osnovnih nastavitev, kot so spreminjanje gesla in ostalih osnovnih podatkov. Na voljo imamo tudi opcijo za dodajanje merilnih sistemov Red Pitaya, ki so nam na voljo, in opcijo za dodajanje uporabnikov. Zadnji dve sta nam na voljo, če smo v skupini administratorjev. Ideja je, da lahko upravitelj sistema (recimo profesor) dodaja študente kot uporabnike, poleg tega pa še merilne sisteme, na katerih so aplikacije v obliki vaj. Študent dobi uporabniško ime in geslo, s katerima dostopa do sistema RAS. Zadnji oddelek prikazuje možnost spreminjanja uporabniških nastavitev.



Slika 4.15: Primer razdelka settings (nastavitve) v RAS sistemu

Poglavje 5

Implementacija preproste aplikacije

V sklopu diplomske naloge smo implementirali tudi preprosto aplikacijo. Hoteli smo pokazati, kako na napravi Red Pitaya zgraditi oziroma prirediti že obstoječe aplikacije za naše potrebe. Delovanje spletnih aplikacij smo že opisali v poglavju 3.2.2. Predstavili smo način, kako uporabniški vmesnik komunicira s programskim krmilnikom trenutno aktivne aplikacije. Paket JSON, ki se pošilja med tema dvema enotama, je zgrajen iz signalov in iz parametrov. Vse potrebne nastavitve, ki jih lahko uporabnik spreminja, je potrebno torej implementirati v obliki parametrov. Na praktičnem primeru si poglejmo izdelavo preproste testne spletne aplikacije za vklop LED diod in shranjevanje podatkov na RAS.

Najprej je potrebno definirati, kaj bi sploh radi z aplikacijo prikazali. Red Pitaya je kickstarter podjetje, ki šele razvija funkcionalnosti, ki bi uporabnikom olajšale delo oziroma izdelavo svojih aplikacij, zato smo tukaj poskušali ohraniti sistem čimbolj preprost. Odločili smo se, da bomo v naši aplikaciji dodali možnost vklopa ter izklopa LED diod in shranjevanje podatkov na RAS.

Prižig LED diod se sliši kot preprosto opravilo, ampak zaradi kompleksnosti samega sistema terja od uporabnika kar precej napora. Za ogrodje

bomo uporabili osciloskop - zastonjsko aplikacijo na Red Pitayi. Celoten komunikacijski sistem in vse potrebne tehnologije, ki jih potrebuje spletna aplikacija za pravilno delovanje, so tukaj že definirane.

Najprej si poglejmo tipično zgradbo scope+gen (osciloskop in generator) aplikacije, ki jo bomo priredili za naše potrebe. Celoten scope+gen direktorij bomo kopirali v nov direktorij, imenovan test_app. V test_app/info/info.json datoteki bomo popravili opis in ime same aplikacije. Ti podatki se nam prikažejo na osnovni strani naprave Red Pitaya, ko poskušamo aplikacijo zagnati. V poglavju 3.3 smo si pogledali osnovno arhitekturo Red Pitaya aplikacije. Med drugim smo omenili, da se v src direktorju nahaja jedro same aplikacije. Spodnji seznam prikazuje najpomembnejše datoteke, ki se nahajajo v src direktorju.

- main.c
- fpga.c
- fpga_awg.c
- worker.c
- generate.c

Poleg zgornjih datotek se v direktoriju nahajajo tudi druge datoteke in njim pripadajoči ustrezne definicijske datoteke (angl. header), ki pa za nas niso tako pomembne, zato smo jih izpustili. Osnovne nastavitve naprave Red Pitaya so opisane v datoteki main.c. Tam se zgodi tudi zagon vseh potrebnih modulov. Datoteki fpga.c ter fpga_awg.c skrbita za komunikacijo z FPGA, sam worker.c pa vsebuje logiko za neprekinjeno delovanje sistema. Lahko si predstavljamo, da je worker.c obsežna nit, ki konstantno izvaja ene in iste funkcije vse dokler ne pride do prekinitve. V main.c datoteki imamo definiran seznam C struktur tipa "rp_app_params_s", ki opisujejo zgradbo vsakega parametra. "Rp_app_params_s struktura"vsebuje naslednje elemente:

- ime

- vrednost
- fpga_update
- read_only
- min
- max

Za vsako funkcionalnost, ki jo bomo predstavili, je potrebno na seznam struktur, ki se nahajo v datoteki main, dodati nov parameter. Dodali bomo štiri parametre: *led_diode*, *set_led_diode*, *save_data* in *channel_number*.

5.1 Kontrola LED diod

Oglejmo si najprej implementacijo prižiga LED diod. V seznam *rp_app_params_s* v datoteki main.c najprej dodamo parametra za LED diode. Parameter mora biti zgrajen v skladu z zgoraj opisano strukturo.

```
{  "led_diode", 1, 0, 0, 1, 8 },
{  "set_led_diode", 0, 0, 0, 0, 2 },
```

V prvem smo nastavili začetno vrednost na 1, minimalno vrednost na 1 in maksimalno vrednost na 8. Tako lahko uporabnik prižiga LED diode od 1 do 8. Diodo 0 smo izpustili, ker služi drugim namenom. Parameter set_led_diode ima začetno vrednost 0, minimalno 0, maksimalno pa 2. Uporabili ga bomo za signal, kdaj uporabnik pritisne na gumb za vklop LED diode(1), kdaj na gumb ugasni LED diodo(2) in za primer, ko ni pritisnjen noben gumb(0). V main.h datoteko je potrebno dodati še primeren C makro, ki opisuje vsak parameter.

```
#define LED_DIODE             44
#define SET_LED_DIODE         45
```

Številka, ki pripada definiciji, označuje mesto v strukturi, kjer je bil parameter definiran in se mora z njo tudi ujemati.

Pri naslednjemu koraku je potrebno dodati FPGA preslikovanje t.i. Housekeeping spominskega bloka, kjer se nahaja register z LED diodami. Ta blok vsebuje nekatere osnovne podatke o Red Pitayi, tako kot register v katerem lahko nastavljamo stanje LED diod, zato ga bomo potrebovali pri naši aplikaciji. Kako se fizične naslove preslika v logične oziroma kako vrednost kazalca C strukture nastaviti mesto, kjer se začne želen spominski blok, smo pokazali že v poglavju z 4.3. Tukaj bomo zato opisali le nekaj dodatnih zahtev, ki so potrebne, da lahko pridobljene podatke s preslikovanja vključimo v Red Pitaya aplikacijo. V src direktorju ustvarimo datoteko *house_kp.c* ter *housekeep.h*, znotraj katerih definiramo metodi *housekeep_init()* ter *housekeep_release()*, ki skrbita za preslikovanje fizičnih naslovov v C strukture. Dodali bomo še dve funkciji: za prižiganje in ugašanje LED diod.

```
int set_led(int led){
        house_reg->led_control |= 1 << led;
        return 0;
}

int unset_led(int led){
        house_reg->led_control &= ~(1 << led);
        return 0;
}
```

Sedaj je potrebno samo še pravilno vključiti housekeep.h v main.c datoteko. To naredimo v datoteki main.c s C ukazom

```
#include "housekeep.h"
```

Sedaj, ko imamo vse na razpolago, je potrebno v pravih funkcijah tako inicializirati kot sprostiti uporabljene vire t.i. Housekeeping spominskega bloka. V metodi *rp_app_init* bomo dodali naslednje vrstice:

```
if(housekeep_init() < 0){
        return -1;
}
```

V metodi rp_app_exit pa naslednje:

```
housekeep_release();
```

Zgornja metoda ne sme nikoli spodleteti, zato ne preverjamo, ali se je izvršila pravilno ali ne. Sedaj, ko imamo vse pravilno vključeno, dodamo na ustrezno mesto v aplikaciji, kjer se ob vsaki iteraciji preverjajo parametri, še pogojni (if) stavek za preverjanje, ali je uporabnik kliknil na gumb za prižiganje/ugašanje LED diod.

```
if(rp_main_params[SET_LED_DODE].value == 2){
        unset_led(rp_main_params[LED_DIODE].value);
        rp_main_params[SET_LED_DODE].value = 0;
}
```

Sedaj smo implementirali vse potrebne funkcionalnosti. Uporabimo Makefile, ki nam je na voljo, in projekt prevedemo. V Makefile datoteki smo spremenili vrstico, kjer se dodajo objekti na seznam za prevajanje s tem, da smo uporabili wildcard ključno besedo za vse .c datoteke. Tako se projekt prevede in poveže vse .c datoteke, ki se nahajajo v direktorju in jih ni potrebno ročno dodajati.

Zadnja stvar, ki nas čaka, je modifikacija uporabniškega vmesnika. V podrobnosti programiranja grafičnih elementov se ne bomo spuščali. Potrebno je le omeniti, da se ob vsaki iteraciji parametri v uporabniškem vmesniku (index.html datoteki) shranijo v *params.local* ter *params.orig* spremenljivki. Ob spremembi nastavitev s strani uporabnika vmesnik nato preveri, ali se spremenljivki med seboj razlikujeta in nato pokliče metodo sendParams(), ki pošlje sveže parametre na napravo Red Pitaya. Preverjanje enakosti obeh *param* spremenljivk opravi uporabniški vmesnik, zato je v nalogi nismo opisali. V naši testni aplikaciji smo uporabili vnosno polje za izbiro diode in gumb za vklop (in prav tako za izklop). Spodnje vrstice kode prikazujejo pošiljanje podatka na napravo Red Pitaya.

```
function changeLed(state) {
        params.local.set_led = \
                parseLocalFloat(\$('#led_on').val());
    params.local.change_led = 1;
    sendParams();
}
```

Funcija *parseLocalFloat* spremeni numerične podatke v tako obliko, ki jo naprava Red Pitaya lahko sprocesira.

Za pravilno delovanje aplikacije je potrebno popraviti še ime same aplikacije v spremenljivki *app_id*, ki smo ga definirali kot ime direktorija same aplikacije (test_app). Preko tega imena se zgodi POST request na Nginx strežnik, ki napravi Red Pitaya pove, katero aplikacijo mora naložiti v spomin.

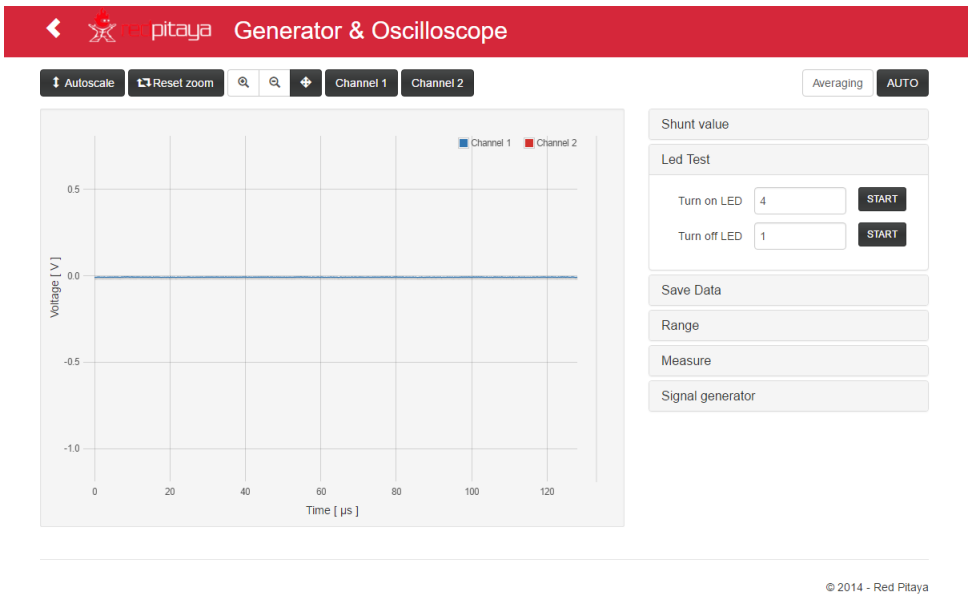
```
var app_id = 'test_app';
```

Sedaj je potrebno celotno prevedeno aplikacijo z uporabo ukaza scp poslati na merilni sistem Red Pitaya. Spletne aplikacije se nahajo v direktoriju /opt/redpitaya/www/apps/. Treba je omeniti, da je Red Pitaya sistem zaradi morebitnih motečih dejavnikov primarno nastavljen samo na bralni dostop. Za pravilno kopiranje datotek se je zato potrebno najprej prijaviti na napravo in pognati ukaz *RW* in tako spraviti sistem v bralno pisalni (angl. Read-Write) način.

Ob uspešnem pošiljanju aplikacije na merilni sistem uporabimo IP naslov, preko katerega dostopamo do osnovne strani naše naprave Red Pitaya. Med aplikacijami vidimo novo Test Application aplikacijo.

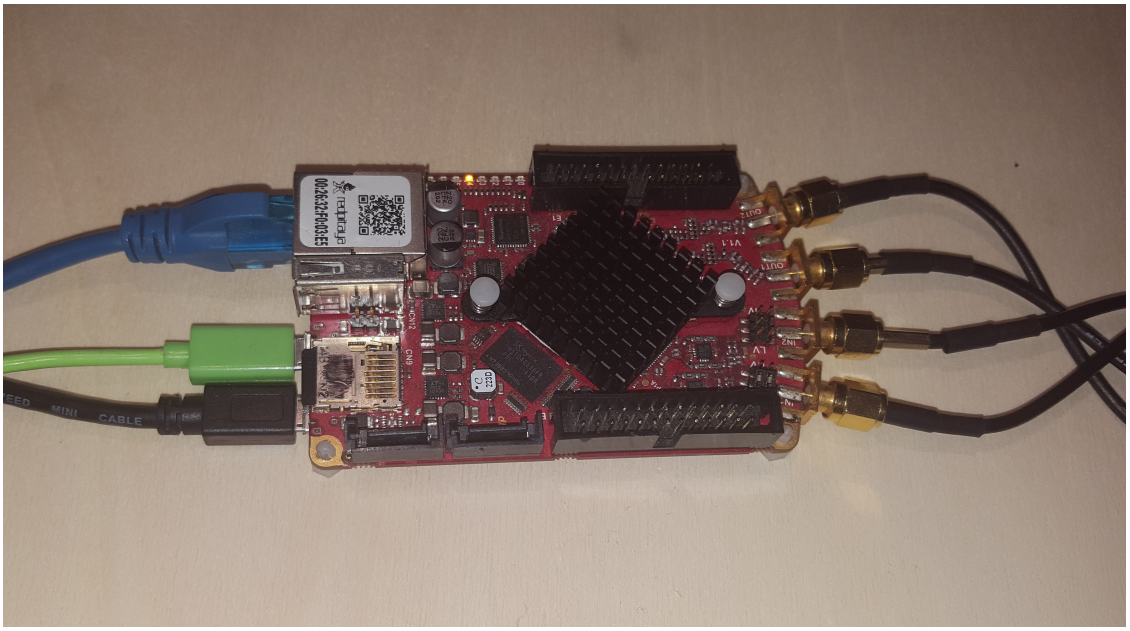
Slika 5.1: Vstopna spletna stran naprave Red Pitaya

Ob kliku na ikono test aplikacije (same ikone nismo spreminjali, zato še vedno kaže GEN, kar je ikona scope+gen aplikacije) se nam zažene testna aplikacija, v kateri lahko preizkusimo delovanje funkcionalnosti za vklop ter izklop LED diod.



Slika 5.2: Uporabniški vmesnik testne aplikacije

Ob kliku na START gumb v oddelku Turn on LED se nam na napravi Red Pitaya vklopi dioda številka 4.



Slika 5.3: Naprava Red Pitaya s prižgano LED diodo številka 4.

Vidimo lahko, da je na napravi Red Pitaya po kliku na gumb res vklopljena LED dioda številka 4, kar prikazuje slika 5.3.

5.2 Shranjevanje podatkov na sistem RAS

Naslednja stvar, ki smo jo implementirali, je shranjevanje podatkov na RAS. Podobno kot pri vklopu LED diod je tudi tukaj potrebno najprej definirati nove parametre. Ponoviti je potrebno prejšnje korake, le da v tem primeru dodamo parameter z imenom *SAVE_DATA* in parameter *CHANNEL_NUMBER* s spodnjimi vrednosti.

```
{  "save_data", 0, 0, 0, 0, 1 }
{  "channel_number", 1, 0, 0, 0, 3 }
```

Spet prevedemo aplikacijo in preverimo, ali je parameter pravilno nastavljen. Za shranjevanje podatkov bomo uporabljali datoteke. Prvotna ideja je bila,

da bi uporabniški vmesnik kar sam pošiljal podatke na strežnik, ampak se je zaradi prepovedi pošiljanja podatkov na drugo domeno izkazala za nepraktično (RAS sistem teče na drugi domeni kot naša aktivna Red Pitaya). Zato smo se odločili, da bomo poskrbeli za shranjevanje podatkov kar na kontrolerju (oziroma jedru Red Pitaya aplikacije). Uporabnik na uporabniškem vmesniku aplikacije najprej klikne na gumb za shranjevanje podatkov. Vmesnik nato pošlje podatke kontrolerju. V jedru strukture aplikacije, v direktorju /src, smo opisali tudi worker.c datoteko. Ta skrbi za periodično delovanje aplikacije dokler ne pride do prekinitve. Tukaj se tudi pošiljajo naprej na strežnik Nginx podatki, ki jih zajameta fpga in fpga_awg datoteki. Za shranjevanje podatkov je potrebno zajeti signale, preden se pošljejo naprej in jih shraniti na datoteko. V spodnjo metodo

```
int rp_osc_decimate(...);
```

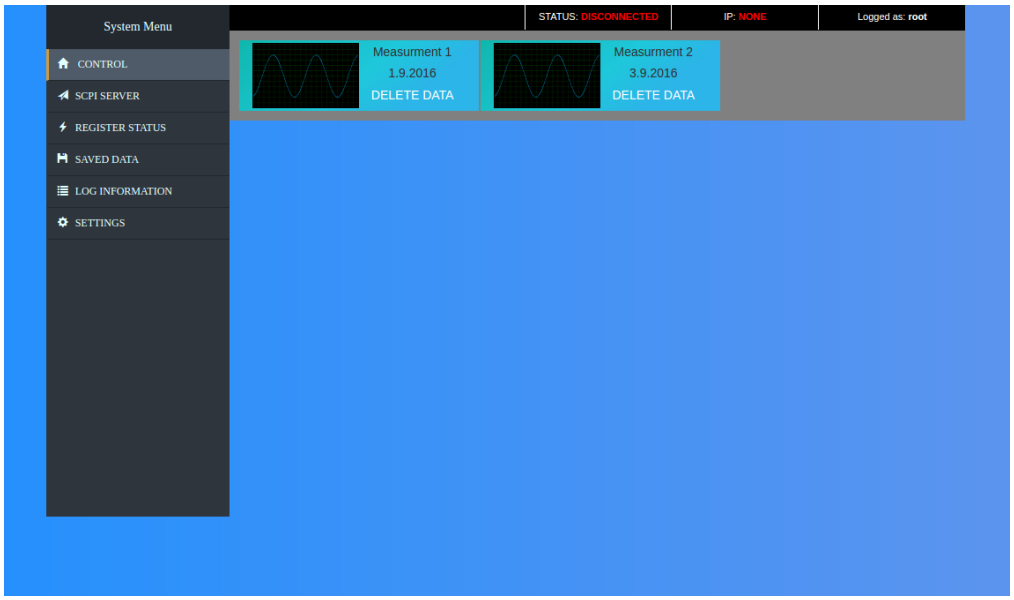
dodamo preverjanje za signal pritiska na gumb na uporabniškem vmesniku na podoben način, kot smo to naredili pri prižigu LED diod. Sama metoda opravi decimacijo oziroma redčenje podatkov in jih pošlje naprej. Uporabi se ob vsakem novem zajemu svežih podatkov; podatki po redčenju so taki, kot jih vidi uporabnik, zato smo lahko prepričani, da shranjujemo prave podatke. Spodnja koda prikazuje preverjanje in zapisovanje signala v datoteko; odvisna pa je od *channel_number* parametra, na podlagi katerega zapiše samo vrednosti prvega ali drugega signala. Če je vrednost parametra enaka 3, zapiše oba.

```
// Save data to file
if((int)save_data == 1){

  // Create files if they do not exist
  FILE *channel1 = fopen("/tmp/channel1", "w+");
  FILE *channel2 = fopen("/tmp/channel2", "w+");

  for (i = 0; i < out_idx; i++) {
    switch((int)channel_number){
          case 1:
            fprintf(channel1, "%f\n", cha_s[i]);
                  break;
          case 2:
            fprintf(channel2, "%f\n", chb_s[i]);
                break;
          case 3:
                fprintf(channel1, "%f\n", cha_s[i]);
                fprintf(channel2, "%f\n", chb_s[i]);
                break;
    }
  }

  fclose(channel1);
  fclose(channel2);
}
```

Spremenljivki *cha_s* ter *chb_s* sta kazalca na vsebino signala kanala A in kanala B. Kazalec se pošlje na strežnik Nginx, kjer se potem servira uporabniškemu vmesniku.

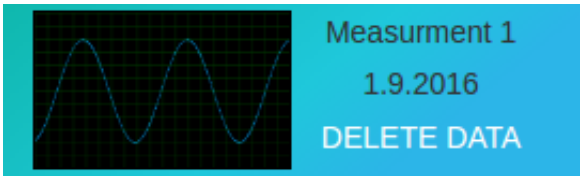
V RAS sistemu uporabimo AngularJS funkcijo, ki periodično izvaja POST zahteve na Flask strežnik. Ob vsaki iteraciji strežnik preveri obstoj datotek channel1 ter channel2, seveda le, če imamo povezan kakšen merilni sistem Red Pitaya. V primeru, da je datoteka polna, prebere podatke in jih najprej shrani v bazo. Za ta namen smo v bazi definirali novo polje, ki ima poleg primarnega ključa datum zajema podatkov, podatke in tuji ključ uporab-

nika, ki je bil prijavljen, ko se je meritev opravila. Novozajeti podatki se shranijo v podatkovno bazo in se pošljejo ter vrnejo kot odgovor na GET zahtevo AngularJS uporabniškemu vmesniku. Medtem strežnik pobriše datoteko iz sistema. AngularJS prikaže podatke na spletni strani RAS sistema. Spodnja slika prikazuje primer dveh shranjenih podatkov. V podatkovnem bloku imamo prikazan ime zajema, datum zajema ter možnost brisanja podatkov iz baze. Podatki so vezani na uporabnika, ki je prijavljen v RAS sistem. Vsak uporabnik ima tako svoje shranjene podatke.



Slika 5.4: Primer strani s shranjenimi podatkih o signalih

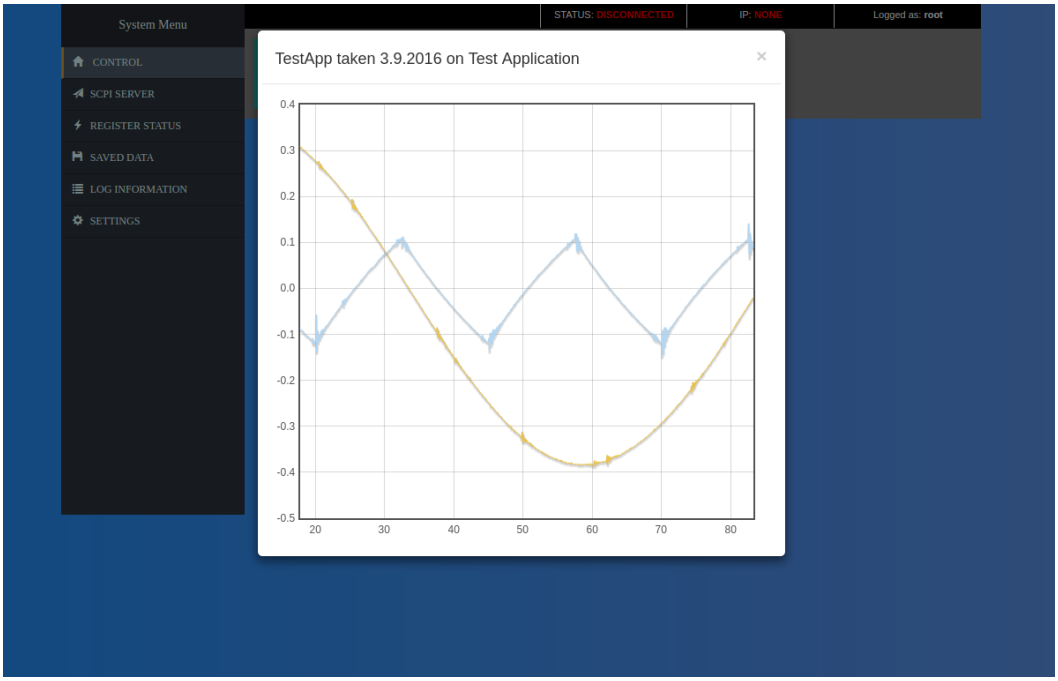
Zgornja slika prikazuje sekcijo shranjenih podatkov, spodnja pa primer bloka, ki vsebuje shranjene podatke.



Slika 5.5: Podatkovni blok, s shranjenimi podatki

Ob kliku na DELETE DATA strežnik izbriše shranjene podatke iz po-

datkovne baze, ob kliku na sliko signala pa se nam pokaže modalno okno, kjer lahko shranjen signal vidimo bolj podrobno. RAS ne omogoča nobenega dela s shranjenimi podatki; to opcijo bi lahko implementirali v prihodnjih iteracijah. Za prikaz podatkov smo uporabili zelo priljubljeno knjižnico flot [22].



Slika 5.6: Primer povečanega shranjenega signala

Poglavje 6

Konfiguriranje RAS sistema

V tem delu bomo govorili o objavi kontrolne aplikacije. Za objavo smo uporabili orodje Nginx, ki smo ga opisali že v poglavju, v katerem smo opisali arhitekturo Red Pitaya aplikacij. Nginx se uporablja za preusmeritev prometa, ki posluša na podani domeni ali vratih. Za primer uporabe si bomo pogledali prenos prometa na lokalnem omrežju. Najprej si je potrebno namestiti Nginx strežnik. Točna in nazorna navodila najdemo na njihovi spletni strani (uporabljamo Ubuntu operacijski sistem). Po namestitvi orodja uporabimo ukaz nano in tako ustvarimo datoteko v direktorju /etc/nginx/sites-avaliable. Pisanje in ustvarjanje datotek v /etc direktoriju zahteva administratorske pravice, zato je potrebno pred ukazom uporabiti še sudo in vpisati administratorsko geslo. V polje, ki se nam odpre, napišemo spodnje vrstice.

```
server {
        listen 80;
        location / {
                proxy_pass http://localhost:5000;
        }
}
```

Datoteko shranimo in jo poimenujemo ras-site. Zgornja koda posluša vse prispele zahteve na naslov http://localhost in ves promet preusmeri na naslov, kjer zahteve pričakuje sistem RAS. Če bi želeli aplikacijo objaviti, bi morali

namesto lokalnega naslova (localhost) uporabiti pravo domeno ali pa zunanji IP naslov.

Ustvarili smo konfiguracijo v sites-avaliable; če pa želimo, da postane veljavna, jo je potrebno kopirati v direktorij /etc/nginx/sites-enabled. To lahko preprosto opravimo s simboličnimi povezavami. Spodnja ukaza ustvarita simbolično povezavo na datoteko ras in hkrati opravita ponovni zagon orodja Nginx, da se konfiguracija upošteva.

```
sudo ln -s /etc/nginx/sites-avaliable/ras /etc/nginx/sites-enables/
systemctl restart nginx
```

Ob pravilni konfiguraciji bi RAS sistem moral biti dosegljiv na naslovu http://localhost/

Poglavje 7

Sklepne ugotovitve

V diplomski nalogi smo si najprej pogledali osnovne lastnosti merilnega sistema Red Pitaya. Predstavili smo samo arhitekturo, osnovne lastnosti in tok podatkov na tipični Red Pitaya aplikaciji. Predstavili smo nekatera pomembna orodja, ki se uporabljajo tako zunaj kot znotraj merilnega instrumenta.

Izdelali smo samostojen sistem za oddaljen dostop (RAS), v katerega smo vgradili nekaj najpomembnejših orodij na Red Pitayi. Uporabili smo osnove spletnega programiranja in tako skušali približati svet vgrajenih sistemov s sodobnim svetom spletnih tehnologij. Sam sistem služi bolj ali manj kot prototip izdelka, ki lahko z dodatnim delom postane razširjen sistem oziroma ogrodje za kontrolo in upravljanje vgrajenih sistemov povsod po svetu.

V sklopu same diplomske naloge smo implementirali tudi preprosto spletno Red Pitaya aplikacijo za shranjevanje podatkov na RAS sistem in vklop ter izklop LED diod. Pogledali smo si osnoven proces gradnje spletnih Red Pitaya aplikacij in dodali osnovno funkcionalnost za prižiganje LED diod. Sam razvoj spletnih aplikacij je še vedno večinoma v razvojni fazi, vseeno pa si z našo diplomsko nalogo lahko pomagamo pri razumevanju tipičnega procesa komunikacije, ki se dogaja med njimi.

Mislim, da smo v diplomskem delu pregledali le vrh ledene gore, del pod vodo pa lahko nudi teme in ideje za prihodnost, v kateri bi lahko iz nastalega

prototipa naredili končen izdelek.

Literatura

[1] ARM architecture (2000) [Online]. Dosegljivo: hhttps://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/third-party/ddi0100e_arm_arm.pdf [Dostopano 6.9.2016].

[2] Arm Cortex A15 Chip (2015, July). [Online]. Dosegljivo: http://www.extremetech.com/wp-content/uploads/2011/10/23565-arm_cortex_a15_super.jpg [Dostopano 5.9.2016].

[3] Red Pitaya logo. [Online]. Dosegljivo: http://www.i-tech.si/announcements/red-pitaya-on-wwwredpitayacom [Dostopano 13.7.2016]

[4] AngularJS documentation (2016, September) [Online]. Dosegljivo: https://angularjs.org/ [Dostopano 14.7.2016]

[5] Flask web-framework documentation (2016, September) [Online]. Dosegljivo: http://flask.pocoo.org/ [Dostopano 14.7.2016]

[6] PostgreSQL [Online]. Dosegljivo: https://www.postgresql.org/ [Dostopano 14.7.2016]

[7] B. W. Kernighan, D. M. Ritchie. "The C programming languge second edition" (1988).

[8] Make (August, 2016) [Online] https://en.wikipedia.org/wiki/Make_(software) [Dostopano 14.7.2016]

[9] Nginx (2016, September) [Online]. Dosegljivo: https://www.nginx.com/resources/wiki/ [Dostopano 14.7.2016]

[10] Zynq-7000 family architecture overview picture. [Online]. Dosegljivo: http://www.xilinx.com/content/xilinx/en/products/silicon-devices /soc/zynq-7000/_jcr_content/imageTabParsys/tab-productAdvantages /xilinxcolumns_3045/column3/xilinxtextmodal_a743/textModalXilinxParsys/ xilinximage_7514.img.jpg/1446507500951.jp [Dostopano 16.7.2016]

[11] Zynq-7000. (2016) [Online]. Dosegljivo: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html [Dostopano 16.7.2016]

[12] Red Pitaya hardware overview. (2016) Dosegljivo: http://wiki.redpitaya.com/tmp/thumb/RedPitaya_HW_overview.png/600px-RedPitaya_HW_overview.png [Dostopano 16.7.2016]

[13] Red Pitaya basic communication overview. (2016, March) [Online]. Dosegljivo: http://wiki.redpitaya.com/index.php?title=OLD_OS_Developer_Guide [Dostopano 20.7.2016]

[14] Red Pitaya terminal console example. (2016, August) [Online]. Dosegljivo: http://wiki.redpitaya.com/index.php?title=Console_connection [Dostopano 20.7.2016]

[15] Red Pitaya terminal applications diagram. (2016, March) [Online]. http://wiki.redpitaya.com/index.php?title=OLD_OS_Developer_Guide [Dostopano 23.7.2016]

[16] Red Pitaya web applications diagram. (2016, March) [Online]. http://wiki.redpitaya.com/index.php?title=OLD_OS_Developer_Guide [Dostopano 24.7.2016]

[17] IEEE-488 cable. [Online]. Dosegljivo: http://blog.testequipmentconnection.com/wp-content/uploads/2013/09/IEEE488plug1.jpg [Dostopano 28.7.2016]

[18] SCPI-99. [Online]. Dosegljivo: http://www.ivifoundation.org/docs/scpi-99.pd [Dostopano 17.8.2014].

[19] I. Jeras, M. Oblak. "Red Pitaya FPGA memory map", 2015. [Online]. Dosegljivo: http://www.redpitaya.com

[20] GNU General Public Licence. [Online]. Dosegljivo: https://www.gnu.org/copyleft/gpl.html. [Dostopano 20. 9. 2014].